189

Computer Science Department

TECHNICAL REPORT

CONSTRAINT PROPOAGATION ON REAL-VALUED QUANTITIES

By

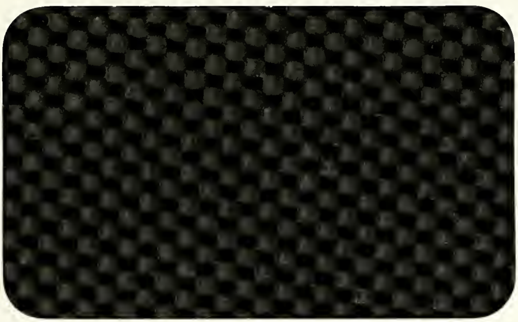
Ernest Davis

November 1985

Technical Report#189

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012



CONSTRAINT PROPOAGATION ON REAL-VALUED QUANTITIES

By

Ernest Davis

November 1985

Technical Report#189

ABSTRACT

Many AI applications require some kinds of reasoning about real-valued quantities. Constraint propagation, of various kinds, is often used to perform this kind of reasoning. In this paper, we focus on a particular kind of constraint propagation, where quantities are labelled with signs or with intervals, and these labels are propagated through recorded constraints. We study both the AI applications and the computational effectiveness of such systems, for various kinds of constraints.

Constraint Propagation on Real-Valued Quantities

Ernest Davis

1. Introduction: Simple Quantity Knowledge Bases

Much of the knowledge in many AI reasoning systems can be expressed as mathematical relations on real-valued quantities. For example, temporal reasoning involves relations on times and durations; spatial reasoning involves relations on lengths and angles; physical reasoning involves relations on masses, temperatures, energies, etc. Frequently, the maintenance of these relationships and the performing of inferences on them can be separated into a distinct module called a *quantity knowledge base*. A quantity knowledge base interacts with the rest of the system's inference mechanism in two ways. The system provides the quantity knowledge base with mathematical relations, either perceived or derived from non-mathematical inferences; and the quantity knowledge base uses these to infer new relations which it can report to the rest of the system.

For many applications, it is possible to restrict the nature of facts and inferences in the quantity knowledge base to a particular kind, which we will call "simple" quantitative inference. In simple inference, the program is given a set of facts about particular quantities, and it is asked to use these facts to deduce the value of some quantity. Typically, the input facts do not constrain the queried quantity to have a single value, so the program will return a range of possible values. For example, we might tell the program, "Joe is younger than Mary and older than Sue. Mary is between forty and fifty. Sue is less than twenty years younger than Mary." If we now ask, "How many years old is Joe?" the system should answer, "Between twenty and fifty."

Formally, we can characterize the problem here as follows: We are given a conjunction of variable-free clauses, and we are asked for the range of values possible for a given term if all the formulas are satisfied. To get a better grip on the scope of this problem, it is best to consider what is excluded here:

1) Individuals can only be real-valued quantities. It is not allowable to use predicates or functions whose arguments are real-valued functions or sets of real numbers.

2) Input facts must not have any quantifiers. A fact like "Jim is older than Mary" is acceptable. A fact like "For all values of t, $vt-1/2gt^2$ (the height of a particular ball at time t) is less than the height of the wall," is not acceptable. Note that, if the set of quantities is finite, a general rule can be incorporated by asserting it separately for each quantity involved. For example, the electronics rule "Voltage = Current × Resistance" can be asserted separately of the quantities associated with each resistor at some fixed instant in the system. Similarly, if we have isolated a number of heights of the ball at different instants as significant, then we can assert that each of them is lower than the wall. However, there is no way that such a finite treatment will support the deduction that the ball is *never* higher than the wall.

3) Queries must have the form, "What is the value (or possible set of values) for some particular term?" Note that consistency queries of the form "Is p true?" can be considered as a special case, by treating the proposition p as a boolean term. For example, the queries, "Is the bath water scalding?" or "How long did the 30 years war last?" are acceptable. "Is the height of the wall greater than all values of $vt-1/2gt^2$?" is not. Another kind of query which is unacceptable is the *individual retrieval* query, "What known quantity lies in such-and-such range?" (Systems which must answer individual retrieval queries generally use some kind of discrimination tree. [McDermott and Davis, 84])

Many different kinds of AI systems use simple quantitative inference in various ways. For example, the ACRONYM system for visual recognition [Brooks, 81] categorizes a perceived object, such as an airplane, as belonging to a given type, by extracting constraints on the object's geometric parameters from a vision system, and checking that these constraints are consistent with the known constraints for that type. Physical reasoning systems, such as QPT [Forbus, 85], CONSTRAINTS [Sussman and Steele, 80], and many others, use constraints on physical parameters to determine the state of a physical system. (The use in these systems of qualitative differential equations to predict future system states is not simple inference, and it is outside the scope of this paper.)

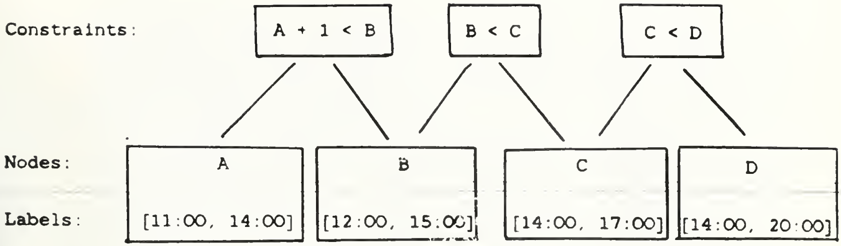
AI systems differ from many other systems which do mathematical computations in that accuracy is relatively unimportant, while robustness under limitations of time and data is very important. Generally, the objective in an AI system is that obvious, important inferences be performed quickly. The general, qualitative aspects of a situation should be readily available; exact numeric values can often be left to a specialized system. We can live with a fairly frequent rate of error and uncertainty on the part of the system, even for problems which are in principle calculable, as long as the system behave correctly on problems which are intuitively clear-cut. Speed, however, is of the essence, given that AI knowledge bases tend to be large and that practical AI systems are often constrained to work in real time. Equally important is the ability to give partial answers given limited information, since the input to AI systems is usually incomplete and imprecise.

2. Networks of constraints.

A very common structure for simple quantitative knowledge bases is as a constraint network. The *nodes* of a constraint network are quantities or terms on quantities. Each node has associated with it a unary predicate, or set, called its *label*, which characterizes the range of values possible for its term. Nodes are connected by *constraints*, which express the relations between these terms which have been input or inferred by some other module. Nodes may also be connected by *implicit constraints*. These are mathematical identities or general laws which true by virtue of the structure of the terms or by the nature of the quantity space. Implicit constraints are represented procedurally rather than declaratively. There are pointers from the constraints to the nodes, and vice versa. A constraint network is thus a kind of semantic network for quantities.

For example, in a planning system, we might have nodes representing event endpoints like BEGIN(PAINT_LADDER), END(PAINT_LADDER), BEGIN(PAINT_CEILING), END(PAINT_CEILING); constraints representing planning rules like BEGIN(PAINT_CEILING) + 1 hr. < END(PAINT_CEILING) or END(PAINT_CEILING) < BEGIN(PAINT_LADDER); and labels which bound the absolute time, like BEGIN(PAINT_CEILING) $\in$ [11:00 A.M., 2:00 P.M.]. We might also choose to include event durations (in duration space) as another kind of node, like DURATION(PAINT_CEILING), with labels representing the time necessary for the event, like DURATION(PAINT_CEILING) > 1 hr. In this case, the general rule "forall (X) BEGIN(X) + DURATION(X) = END(X)" would probably be represented as an implicit rather than an explicit constraint. (Figure 1)

Systems which operate on constraint networks fall into three classes. In a *constraint inference* system, an inference engine combines constraints to deduce new constraints, and answers queries by finding or deducing an appropriate constraint. In a *label inference* system, all inferences are mediated through the labels, and no new constraints are ever created, except through input. Constraints are used to calculate new labels, and queries are answered using the current labels. In a *non-inferential* system, some specialized mathematical technique is used whose intermediate steps do not take the form of deducing true statements. For example, solving a set of linear equations by finding the inverse of the coefficient matrix and multiplying is non-inferential. This paper deals primarily with label inference systems, though other systems will be mentioned in passing.



Interpretation: A = BEGIN(PAINT_LADDER)

B = END(PAINT_LADDER)

C = BEGIN(PAINT_CEILING)

D = END(PAINT_CEILING)

Figure 1

A Constraint Network

Constraint systems are also divided into "wholesale" and "incremental" systems. In a wholesale system, all input constraints are available at the beginning of processing, and are fixed thereafter. In an incremental system, accepting new constraints alternates with answering queries. (In algorithm theory, these are called "off-line" and "on-line" systems, respectively.) In AI contexts, wholesale systems are generally used for relatively small sets of constraints, such as those which describe a simple physical situation. Incremental systems are generally used for comparatively large systems, such as timelines, or spatial maps. Also, systems which alternate between strictly quantitative inference (inference which involves only quantities and constraints on quantities) and non-quantitative inferences which interact with quantitative information, such as physical inferences, are best considered as incremental systems, as far as the quantity knowledge base is concerned. In wholesale systems, we are concerned with the time to accept a set of constraints, and we are willing to accept running times like $O(n^2)$ or $O(n^3)$. In incremental systems, we are concerned with the time to add a single constraints, and we look for running times which stay small as the system gets large, such as $O(\log(n))$.

Note that the lines between label inference, constraint inference, and non-inferential systems are somewhat hazy. Any constraint inference system can be viewed as a label inference system, by considering a constraint to be a boolean term and an inference to be a propagation of a boolean value. For example, the inference "$X<2$" from "$X^2<4$" can be viewed as the propagation of the label "TRUE" from the boolean term "$(X^2<4)$" to the boolean term "$(X<2)$". Also, all inference systems rely on non-inferential techniques to carry out basic steps; we do not do addition using the Peano axioms. On the other hand, all non-inferential techniques can be viewed as inferential, if we are willing to accept sufficiently strange propositions. Any computation on a Turing machine can be viewed as a process of inference, where the inference steps are "The first state of the machine is ...", "The second state of the machine is ..." etc.

The chief questions in evaluating a simple quantity inference system are the following:

a) What kind of constraints will the system accept? What kinds of queries will it answer? What kinds of answers does it give?

b) What AI problems can be formulated within the scope of (a)?

c) Is the system wholesale or incremental?

d) How much time is required to accept a constraint or a set of constraints? How much time is required to answer a query?

e) Is the system complete? A system is *complete* if the range of value returned for a queried term is exactly the set of values consistent with the constraint. It is complete up to the query-answer language if the answer to a query characterizes the set of possible values as tightly as is possible in the language.

f) What does computation theory say about the requirements of a complete algorithm to answer these queries from these constraints? How does the system compare to the theoretical ideal?

3. Label Inference Systems

Label inference systems have two basic operations: *query answering*, in which labels on nodes are used to evaluate a given term, and *assimilation*, in which constraints are used to improve the labels. In query answering, the term to be evaluated is expressed as a function of some of the nodes, and the system must calculate the range of value of the term given the label sets of the component nodes. In assimilation, the basic operation is *label refinement*, in which the label set of one node is restricted, based on a constraint and on the labels of all the other nodes in the constraint. For example, if we have the constraint "$X+Y=Z$", and we have the label sets "$X\geq 3$, $Y=1$", then we can deduce that $Z\geq 4$, and add this to the description of the label set of Z. The general form of refinement can be expressed in the following definition:

Definition: Let C be a constraint on nodes $X_1 \ldots X_k$. Let S_i be the label set for X_i. Then

$$\text{REFINE}(C,X_j) = S_j \bigcap \{a_j \mid \text{exists } (a_i \in S_i,\ i=1 \cdots k, i \neq j)\ C(a_1 \cdots a_j \cdots a_k)\}$$

That is, REFINE(C,X_j) is the set of values for X_j which is consistent with the constraint C and with all the labels S_i. A value a_j is in REFINE(C,X_j) if a_j is in S_j and it is part of some k-tuple $a_1, \cdots ,a_k$ which satisfies C and all the S_i.

Note that refinement is sound deductively. Any k-tuple satisfying the constraint C and all the labels S_i must have its jth element inside the refined label.

The Waltz algorithm [Waltz, 75] is the most thorough use of refinement.* It entails using refinement on each constraint and each node over and over until refinement produces no more changes. When this stage is reached, the network is said to have *reached quiescence*. Since refinement is sound deductively, so is the Waltz algorithm, which simply iterates refinement. That is, if a given assignment of values to the quantities satisfies all the constraints and all the starting labels, then it will satisfy the labels calculated by the Waltz algorithm. If the algorithm halts with assigning some parameter the null set, then the input state was inconsistent. Algorithm 1, modified from [Mackworth and Freuder, 85], is an efficient implementation of the Waltz algorithm. Figure 2 shows an example of the Waltz algorithm in action.

The running time of the Waltz algorithm has been extensively analysed in the case where the range of the parameters is a finite set (see [Mackworth, 77], [Freuder, 78], [Mackworth and Freuder, 85]). In particular, Mackworth and Freuder have shown that the algorithm halts after $O(ae)$ calls to REVISE where a is the number of possible values per parameter, and e is the number of constraints. However, this halting result does not apply when the set of possible labels is infinite, as is often the case with quantitative labels. In such cases, the analysis depends critically on the nature of the constraints involved.

The restriction that all information pass from the constraints to the queries via the label sets makes it almost inevitable that some information will be lost. For example, if we start with two nodes A and B labelled "$A \in \{1,2,3\}$" and "$B \in \{2,3,4\}$" and the constraint "$A=B$", then the Waltz algorithm will deduce the new labels "$A \in \{2,3\}$", "$B \in \{2,3\}$", and stop. If we now pose the query "What is the value of the vector $<A, B>$", the system will answer that there are four possibilities: $<2,2>$, $<2,3>$, $<3,2>$, and $<3,3>$, since these are all consistent with the labels, though not with the constraint. Similarly, if we query the system "Is A equal to B", the best answer we can get is "Possibly yes, or possibly not."

If problems of this kind often cause serious harm in the inference system, then the system designer should consider adding new kinds of terms as nodes. For instance, the above problem can be fixed by adding terms of the form $x-y$ as nodes, and using the constraint to deduce the labelling $A-B=0$. If the query answerer now consults this new labelling, it can answer both of the above questions accurately. There is a cost, however, in terms of increased complexity both of assimilation and of query answering. (See section 9).

An alternative approach, pursued in an early version of the SPAM program [McDermott, 1980] is to use labels such that any selection of values from the labels satisfies the constraints. (McDermott calls these "conservative" labels.) That is, the cross product of the labels is a subset of the solution set of the constraints, rather than being a superset, as in systems which use refinement. In the above example, we would use either the set of labels $A \in \{2\}$, $B \in \{2\}$, or the set of labels $A \in \{3\}$, $B \in \{3\}$. If we had the starting labels $A \in \{1,2,3\}$, $B \in \{2,3,4\}$, and the constraint $A \neq B$ then there would be four possibilities for reasonable conservative labellings: $A \in \{1\}$, $B \in \{2,3,4\}$; $A \in \{1,2\}$, $B \in \{3,4\}$; $A \in \{1,3\}$, $B \in \{2,4\}$; and $A \in \{1,2,3\}$, $B \in \{4\}$. Boggess [Boggess, 79] uses an extreme form of this approach in which each quantity is given a single value such that the quantities satisfy the constraints. Thus, in the second

* I have avoided the use of the phrase "constraint propagation," except in a general sense, since it is used ambiguously to refer to label inference, constraint inference, and the Waltz algorithm.

```
/* The set S_i is the current label set of quantity X_i.

   REVISE refines all the parameters of a given constraint, and

   returns the set of all the parameters whose value was changed. */

procedure REVISE (C(X_1 ··· X_k): constraint with k arguments);
  begin CHANGED ← null;
    for each argument X_i do
      begin S ← REVISE(C,X_i);
          if S is empty then halt (the original constraints were inconsistent) else
          if S ≠ S_i then
            begin S_i←S;
                add X_i to CHANGED
             end
       end
     return CHANGED;
 end;

procedure WALTZ;

 begin Q ← a queue of all constraints;
     while Q is not empty do
       begin remove constraint C from Q
             CHANGED ← REVISE (C);
             for each X_i in CHANGED do
                for each constraint C' ≠ C which has X_i in its domain do
                   add C' to Q;
       end
 end
```

Algorithm 1

Algorithm 1 may be modified to handle the addition of a single constraint to an incremental system by modifying the first line of "WALTZ" to read, "Q ← a queue containing the new constraint."

Suppose we start with the following relations and starting bounds:

Bounds: $x \epsilon [1,10]$, $y \epsilon [3,8]$, $z \epsilon [2,7]$.
Relations: $x+y=z$, $y \leq x$.

This would be implemented in a data structure like the following:

Quantities: X:: Bounds: [1,10]; Constraints: (CON1, CON2)
Y:: Bounds: [3,8]; Constraints: (CON1, CON2)
Z:: Bounds: [2,7]; Constraints: (CON1)
Constraints: CON1:: X + Y = Z
CON2:: Y ≤ X

The algorithm would proceed as follows:

The constraint queue would begin with both constraints (CON1, CON2) on it.

CON1 is popped from the queue.
Since X ≥ 1, Y ≥ 3, CON1 (X+Y=Z) gives Z ≥ 4, so reset the bounds of Z to [4, 7]
Since Z ≤ 7, Y ≥3, CON1 gives X ≤ 4, so reset the bounds of X to [1,4]
Since X and Z has been changed, add CON2 to the queue

CON2 is popped from the queue.
Since X ≤ 4, CON2 (Y < X) gives Y ≤ 4 so reset the bounds of Y to [3,4]
Since Y ≥ 3, CON2 gives X ≥ 3, so reset the bounds of X to [3,4].
Since X and Y have been changed, add CON1 to the queue.

Since X ≥ 3, Y ≥ 3, CON1 gives Z ≥6, so reset the bounds of Z to [6,7].
Since only Z has been changed and Z has no other constraints besides CON1,
nothing is added to the stack

Since the stack is empty, quit.

Figure 2

example, she might use the labellings $A=1$, $B=4$ or $A=3$, $B=2$, etc.

This approach, however, is very problematic. It is not deductively sound; hence it permits inferences not warranted by the constraints and it frequently requires withdrawal of previously assigned labels. It is not unique; there are many different sets of conservative labels. Finally, it is much harder to implement than a refinement algorithm, since there is no analog of the "REFINE" operator which allows one variable to be set at a time. Rather, all variables must be considered together. For instance, if, as above, we start with the labels $A\epsilon\{1,2,3\}$, $B\epsilon\{2,3,4\}$, and we add the constraint $A \neq B$, then we must choose between one of the four conservative labellings mentioned above. Finding these conservative labelling involves assiging A and B simultaneously; they cannot be considered separately. Whichever labelling we choose, we run the risk of being wrong. For example, if we choose $A\epsilon\{1,2\}$, $B\epsilon\{3,4\}$, then we have arbitrarily ruled out the consistent possibility $A=3$, $B=2$. The next constraint that comes in may contradict our conservative labels even if it is consistent with the first constraint; in our example, the constraint $A=3$ would violate the labelling we have chosen. If this happens, the system must do some difficult backtracking.

Label inference techniques can rarely be complete, because of the "narrow bandwidth", so to speak, of the labels as a medium of information from the constraints to the queries. The most we can ask is that the assimilation and query answering processes each be separately complete. Assimilation is complete if the label assigned to a term represents accurately the range of values it can attain given the constraints. That is, if assimilation is complete, than we can consistently assign to any node any value within its label, and then can pick values for all the other terms so that all constraints are satisfied. (If the assimilation technique used is sound, then the labels will be maximal with respect to this property.) If the actual range of values is a set which is not described by any label in the label language, then we can say that assimilation is complete up to the label language if the labels assigned are the best labels possible. Note that, by definition, the refinement operation is always complete for a single constraint; however, as we shall see, this does not imply that the Waltz algorithm is complete for a set of constraints.

Query answering is complete if the range of values returned for the queried term is exactly the range permitted by the label sets. That is, if we query the system for the value of term T, and it returns "T is in set S," then, for any value V in S, we may assign values V_1, ... V_n to the nodes so that all implicit constraints are sastified, and so that, if the nodes have these values, then T will have value V. For example, suppose we have nodes A, B, and $A-B$, with labels {2, 3}, {2, 3} and {0}, respectively, as above, and we query the system whether $A=B$. A query system that looks only at the first two labels will answer "Possibly true; possibly false;" a system which looks at the third label will say "Definitely true." The first system is incomplete, while the second is complete.

If a label inference system supports only trivial queries, which ask for the value of some node term, then having a complete algorithm for assimilation is equivalent to having a complete overall algorithm. Likewise, if the system supports only trivial assimilations, in which the only explicit constraints are equivalent to conjunctions of labels on node terms, then having a complete algorithm for quey answering is equivalent to having a complete overall algorithm. If neither queries nor assimilations are trivial, then the narrow bandwidth of labels makes it very unlikely that a label inference system will be complete overall, even if both assimilation and query answering are seperately complete.

Despite this incompleteness, label inference algorithms have a number of significant advantages in AI systems, even when more complete algorithms are available. They run quite fast. They degrade well under time limitations; interrupting them in the middle gives useful partial results, in the labels already calculated. They are easily coded, and easily extended to new types of constraints because the control structure is independent of the kind of constraint. They are easy to implement in parallel; we may simply associate one processor per constraint, and have it repeatedly perform refinement on all its arguments. They are well suited to incremental systems, because adding a constraint generally has only local

effects, even after propagation; propagation does not affect the whole knowledge base. The local nature of the algorithm ties in well with the locality assumptions which are also made in AI problems, such as the assumption in ENVISION [DeKleer and Brown, 85] that physical effects propagate across connections between components, or the assumption in MERCATOR [Davis, 84] that nearby objects will be seen together. Finally, since label inference algorithms use only inferences of a restricted, simple kind, their behavior is easily understood, both by an external user and by another internal module, such as a data dependency system.

There is a significant difference between AI systems and numerical analysis systems in the relative importance they place on algorithm completeness versus quiescence in finite time. In numerical analysis systems, finite time quiescence is considerably less important than rate of convergence and numerical stability. If an algorithm terminates in finite time, very well; if not, it doesn't much matter as long as it converges to a reasonably accurate answer reasonably fast. AI systems, on the contrary, place much less emphasis on completeness and accuracy. The object of an AI system is to make the easy inferences fast; the hard inferences don't matter so much. The gross character of the solution of a constraint set should be generally be evident after a few refinements. If it is not, then either the program is not well attuned to these kind of constraints or the constraint set is hard, in which case we can live with uncertainty. On the other hand, control structures present a great deal of trouble in AI systems, and it is a great boon if forward inferences, such as label inferences, have a natural stopping point, beyond which no further inference is possible. If we use, instead, an arbitrary condition to halt propagation, such as a time limit, then there is no guarantee that some future inference process won't waste its time continuing the propagation, and getting labels which are more precise but essentially useless.

Label inference is in some ways similar to relaxation labelling ([Hummel and Zucker, 83]). Both systems involve propagating values from one node to another locally, and therefore both systems are very suitable for a massively parallel implementation. Label inference, however, is easier to analyze since it is a deductive process, and since the values associated with the nodes change monotonically (the sets of values always decrease.) For this reason, instability problems and timing problems in parallel implementations, which plague relaxation labelling, do not arise in label inference.

The basic characterisics of a label inference system are the constraint language, the query language, the query answering language (as for quantitative knowledge bases generally), the kinds of terms used as nodes, and the kinds of predicates used as labels. Once these are fixed, programming and evaluating the assimilator involves the following considerations:

a) How can an input constraint be expressed as a constraint on node terms?

b) How can refinement be performed on explicit constraints?

c) What kinds of implicit constraints are used, and how can refinement be performed on them?

d) Under what circumstances does the Waltz algorithm reach quiescence? If it does not, how should it be halted?

e) In what order should the Waltz algorithm choose constraints and nodes to refine?

f) Under what circumstances are nodes introduced and removed from the system?

g) Is the assimilator complete?

Designing the query answerer involves the following problems:

h) How can the query be expressed as a function of the terms?

i) How is the query term evaluated?

j) Is the query answerer complete?

4. Label Languages

We will now consider varieties of constraint network systems. As stated above, there are five essential determinants of a system: the types of node terms, the language of labels, the language of constraints, the language of queries, and the language of answers. However, it turns out that the languages of labels and constraints are key for categorization. (For the present, we will assume that all nodes are simple quantities; we will consider more complex nodes in section 9. The language of queries is almost always equal either to the label language or to the constraint language. The language of query answers is typically the same as the language of term labels.) We will therefore begin by separately considering vareties of label language and constraint language, and then describe how they combine.

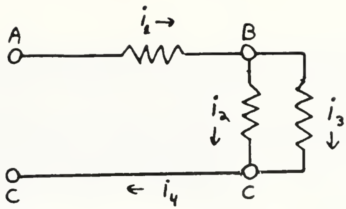
There are two main label languages in use: signs and intervals. Labelling by signs involves marking each node as positive, negative, zero, or indeterminate (denoted POS, NEG, 0, and IND).* This crude categorization often turns out to be sufficient for particular applications, such as naive reasoning about physics ([DeKleer and Brown, 85], [Forbus, 85]). It has been used particularly to characterize variables which represent change. Often the amount of a change is relatively unimportant; what really matters is whether the change is an increment, a decrement, or a constancy. For example, in ENVISION [DeKleer and Brown, 85], the behavior of an electronic system at a particular moment is characterized by the direction of change of each of the voltages and currents. These directions are interrelated by the constraints associated with the network structure. (Figure 3)

Labelling by intervals involves recording for each term a lower and upper bound on the values it can attain. It is often useful in spaces where sign labelling is too coarse a measurement, or which have no absolute zero. Intervals have a number of valuable properties as a language of labels. Intervals are flexible enough to represent both an approximate value and a degree of uncertainty. An interval can be represented by two real numbers and two booleans (to indicate closedness or openness at each end.) All connected sets of real numbers are intervals, and vice versa. Hence, the image of an interval, or a collection of intervals, under a continuous function is itself an interval. Likewise, the intersection of two intervals is an interval. Finally, it is generally reasonably easy to work out the refinement operator for intervals in particular cases.

Two questions are often raised with respect to intervals. The first is whether to use open, closed, or half-open intervals. Clearly, for maximal flexibility, one would want to use all of these. However, in practice, this tends to very much complicate the case analysis for very little practical gain, and it is better to stick either to closed intervals or to open intervals. As between closed vs. open intervals, closed intervals are probably preferable, for three reasons. Firstly, given any bounded set of values, there is a unique minimal closed set which contains it; hence, a best labelling. Secondly, exact values are a special case of closed intervals so exact values and interval values can be handled uniformly. However, the programmer who wishes to exploit this fact must be somewhat careful. He will find that code which is correct and works perfectly well for intervals of finite length may bomb on point intervals because of round-off error. Finally, the image of a closed bounded interval under a continuous function is a closed bounded interval; this is not true of open intervals. However, this advantage is lost if the system also uses intervals which are infinite on the right or the left, as most systems do. Closed intervals are also typically used in the theory of interval analysis [Alefeld and Herzberger, 83].

The other question that is often raised with regards to intervals is that of psychological plausibility. Is it reasonable to treat quantitative knowledge as a step function, to say that one is sure that the length of a stick is between 5.11 and 6.34 feet, and nothing more? Do not

* Strictly speaking, we should, perhaps, include the labels NONNEG(x) ($x \geq 0$), NONPOS(x) ($x \leq 0$) and NONZERO(x) ($x \neq 0$). This would allow refinements such as NONZERO(x,y) implies NONZERO(x), NONZERO(y). However, the slight increase in completeness does not seem to be worth the substantial increase in complexity.



Kirchoff's Voltage Law:

OPP_SIGNS $(\delta(V_{AB}),\delta(V_{BC}),\delta(V_{CA}))$

Kirchoff's Current Law:

$\delta(i_1)=\delta(i_4)$

OPP_SIGNS $(-\delta(i_1),\delta(i_2),\delta(i_3))$

OPP_SIGNS $(\delta(i_2),\delta(i_3),-\delta(i_4))$

Ohm's Law:

$\delta(i_1)=\delta(V_{AB})$

$\delta(i_2)=\delta(V_{BC})$

$\delta(i_3)=\delta(V_{BC})$

Notation:

V_{AB} = Voltage from A to B

i_k = Current in link k

$\delta(x)$ = Direction of change of x (POS, NEG or ZERO)

OPP_SIGNS (x,y,z) = either x,y,z are all zero

or at least one is positive and at least one is negative.

Figure 3

The analysis of a circuit in ENVISION [DeKleer and Brown, 85]

one's beliefs correspond more to a probability distribution with a likelihood gradually diminishing on either side of a central value? There is something plausible about these arguments; unfortunately, the alternatives to intervals are worse. Probability functions require more arbitrary assumptions and introducing more fudge numbers than simple intervals; their semantics is infinitely less clear; and, if done right, they require much more complex computations. Representing uncertain values by (say) a Gaussian probability curve is no less arbitrary that representing them by fixed bounds; and it is very difficult to define what the probability means and how to combine different curves. My feeling is that it is better to pick some safe bounds, to do computations with them, and then to introduce probability at the end by treating the results with a grain of salt. If the computation tells you that the length of the stick is between 5.11 and 6.34 feet long, this should be read as "About 5.7 ± 0.6". If some rule requires that the stick be at most 6.5 feet long, the system should not consider that the condition has been met with any great certainty.

5. Constraint Languages

There are seven major classes of constraint languages which have emerged so far as important in AI systems. In increasing order of complexity these are:

a) Unary predicates

b) Order languages, consisting simply of the order relationship.

c) Systems of equations of the form "$x-y>c$" or "$x-y \geq c$".

d) Linear equations and inequalities with unit coefficients (i.e. all coefficients are -1, 0, or 1).

e) Linear equations and inequalities with arbitrary coefficients.

f) Boolean combinations of constraints.

g) Algebraic equations

h) Transcendental equations.

In this section, we will discuss the AI applications of each of these.

The simplest kind of constraints are unary predicates, i.e. labels. We have already looked at two possible classes of unary predicates: signs and intervals. Other predicates which are often used include exact values and fixed ranges. Exact values are used in systems where quantities are known without uncertainty. This was the practice in old style blocks-world programs, such as SHRDLU [Winograd, 71], which represented the positions of blocks by the exact coordinates of the front lower left-hand corner. Fixed ranges are used in measure spaces which are naturally divided into a finite set of exhaustive, disjoint intervals. For example, the measure space "Water temperature" is naturally divided into the ranges $(-\infty, 32)$ (ice), $[32, 32]$ (melting), $(32, 212)$ (liquid), $[212, 212]$ (boiling), $(212, \infty)$ (gas). The space of voltages across a transistor is divided into the cutoff region, the linear region, and the saturation region. Such divisions are extensively used in naive physics systems such as [Forbus, 85] and [DeKleer and Brown, 85].

Order relations arise in systems where the comparative magnitude are the only quantitative relations of interest. This occurs, for example, in systems which plan the order of events without consideration of their time duration, such as NOAH [Sacerdoti, 77] or GPS [Newell and Simon, 72]. Each level in planning in NOAH specifies a partial ordering on the actions in that level. Order relations are also a large part of the quantitative information available in naive physics systems such as [Kuipers, 85], [Forbus, 85], [DeKleer and Brown, 85]. The binary relations on intervals discussed in [Allen, 83] for use in temporal reasoning are essentially combinations of order relations on the interval endpoints.

Inequalities of the form $x-y \geq c$ are useful in dealing with quantity spaces in which the relative values of two quantities may be known well though their absolute quantities are much more uncertain. This happens particularly in trying to place events on a timeline. For example, one might know that Michaelangelo was a older contemporary of Raphael, and that

the end of the Thirty years war occured about thirty years after its beginning, while having only the vaguest notion of either the dates of these events, or the relation of the first pair to the second pair. Such knowledge can be represented in equations like the following:

date (birth (Raphael)) - date (birth (Michaelangelo)) ϵ [0, 30]
date (end (30_years_war)) - date (start (30_years_war)) ϵ [25, 35]
date (birth (Raphael)) - date (start (30_years_war)) ϵ [-100, 200]
date (birth (Michaelangelo)) ϵ [1400, 1600]
date (start (30_years_war)) ϵ [1500, 1700].*

Similarly, in a planning situation, there might be strong constraints relating the end of a planned action to its beginning, but only weak constraints relating the times of two planned actions whose order had not been decided. Such constraints are used in the planning systems of Vere [Vere, 83] and Dean [Dean, 85].

In measure spaces with scalar multiplication, a bound on the quotient of two quantities is often more useful than a bound on their difference. For example, one might know that both your kitchen table and your house were roughly 2 to 1 rectangles without knowing at all precisely the ratio of their sizes. You might then express your knowledge in the inequalities:

length_of (table) / width_of (table) ϵ [1.8, 2.2]
length_of (house) / width_of (house) ϵ [1.5, 2.5]
length_of (house) / length_of (table) ϵ [10.0, 50.0]

A system of bounds on quotients of positive quantities is isomorphic to a system of bounds on differences; the logarithm function is the isomorphism. Systems which combine bounds on differences and bounds on quotients are more difficult, as we will discuss below.

It may seem strange to single out the class of linear equations and inequalities with unit coefficients. These are chiefly important because they are adequate to express conservation laws. Conservation laws assert that the change in a value over time is equal to the sum of its increments minus the sum of its decrements; they are particularly important in commonsense reasoning. For instance, the change in a bank account is the sum of the deposits plus the interest minus the sum of the withdrawals. The change of the amount of liquid in a cup is the total amount poured in minus the total spillage and evaporation. If we are only interested in reasoning about discrete changes, and we can identify all relevant increments and decrements, then we can write any such rule in the form $Q_f - Q_i = \sum Inc_k - \sum Dec_k$.

Linear inequalities with non-unit coefficients are surprisingly uncommon in AI contexts. Their main application so far has been in analysizing the differential behavior of non-linear systems. For example, in studying small perturbations to an ideal gas, which obeys the law $PV=nkT$ (P=pressure, V=volume, T=temperature, n and k are constants), we may use the differential rule $V\frac{dP}{dt}+P\frac{dV}{dt}=nk\frac{dT}{dt}$. For small perturbations from a starting state, V and P may be treated as constants. The ENVISION system [DeKleer and Brown, 85] uses such relations. (Note that, though [Malik and Binford, 83] talks about arbitrary linear inequalities, all of the examples given are either absolute bounds, order relations, or bounded differences.)

Boolean combinations of constraints have been most extensively studied in naive physics [DeKleer and Brown, 85], [Forbus, 85], in electronic circuit design [Stallman and Sussman, 77], and in planning. They take three major forms. Firstly, there are gated constraints of the form "if P then Q" where P and Q are constraints. These arise in electronics and in physics in considering systems which have several states, each with different rules, such as the state of a transistor, or the phase of a quantity of material. Secondly, there are crude

* Here and throughout this paper, the notation $T\epsilon[a,b]$ where T is a term on quantities, may be taken as an abbreviation for $a \leq T \leq b$. In describing the first order language of these systems, it is not necessary either to use set theory or to use intervals as individuals.

characterizations of more complex constraints. For example, in reasoning about an adjustable valve, we know that opening the orifice increases the current for constant pressure, and that raising the pressure increases the current for constant area, though the exact functional relation between the three quantities may be quite complex. We can summarize this information as follows: if δP, δQ, and δA are the changes in pressure, current, and area over a time interval, then either they are all equal to zero or two of the quantities δP, $-\delta Q$, δA, have opposite signs.* Thirdly, in planning, a very common constraint is that the actor can only do one thing at a time. This can be expressed as the disjunction, "Either the end of ACT1 comes before the beginning of ACT2 or *vice versa*."

Polynomial inequalities often arise from combining ratio information with difference information. For example, suppose that I know that painting the table will take somewhat longer than painting the chair, I intend to start the table about ten minutes after finishing the chair, and I want the whole process to take under two hours. Using $T1$, $T2$, $T3$, and $T4$ for the times when I start the chair, end the chair, start the table, and end the table, respectively, I can express these constraints in the three inequalities, $\frac{T4-T3}{T2-T1}\epsilon[2,3]$; $T3-T2\epsilon[8,12]$; $T4-T1<120$. Together, these are a quadratic set of inequalities.

Geometric reasoning generally involves both algebraic and trancendental equations. The key geometric functions are the distance function, which is algebraic, and the trigonometric functions, which are transcendental. (For some purposes, however, one can avoid the direct use of angles, and only use their cosines, which are algebraic functions of the coordinates.) Surprisingly, given their ubiquity in computer science, the exponential and logarithmic functions seem to be much less important in AI systems.

6. Complexity results and standard techniques.

The next question to address is, for each of these kinds of constraints, what is known about the computational complexity of solving the problem, and what standard algorithmic techniques are available. In each case, there are three problems to consider: (i) the problem of evaluating the bounds on a term of a particular kind, given labels on its arguments; (ii) the problem of performing refinement on a constraint of a given kind; and (iii) the problem of solving a set of such constraints.

6.1. Computation Theoretical Bounds

((i) and (ii)) Evaluating order relationships, linear sums, or boolean combinations of these, on labels is straightforward. Evaluating an algebraic expression on a set of labels is NP-hard, even if the expression is no more than fourth-order. (Yemini [Yemini, 79] shows that solving a set of equations of the form $(x_i-x_j)^2+(y_i-y_j)^2=c_{ij}$ is NP-hard. This problem can be reduced to the problem of whether 0 is in the range of $\sum_{i,j}[(x_i-x_j)^2+(y_i-y_j)^2-c_{ij}]^2$.) However, the bounds on an algebraic expression can be computed in exponential space, and hence doubly exponential time [Kozen and Yap, 85]. Evaluating a transcendental function is, in the worst case, uncomputable. (It is uncomputable whether a given transcendental expression is identically equal to zero.) [Richardson, 68]. The same bounds apply to performing refinement, except for boolean combinations. Since boolean combinations include arbitrary conjunctions, solving a single boolean combination is equivalent to solving a system of them. (See below).

(iii) A set of order relationships or bounds on differences may be solved in time n^3. A set of linear relations can be solved using Karmakar's algorithm in time $O(n^{3.5}E^2)$, where n is

* DeKleer [DeKleer and Brown, 85] and others write this relation, "$[\delta P]-[\delta Q]+[\delta A]=0$," where $[X]$ is the sign of X. This notation is confusing, and its semantics are unclear. Under any reasonable definition of addition of signs, POS - POS + POS = IND. Thus POS, POS, POS would not be a solution of the above equation; the only solution would be ZERO, ZERO, ZERO. What is really meant is "$[\delta P]-[\delta Q]+[\delta A]$ contains 0."

the number of variables and E is the size cf the problem. [Karmakar, 84]. Solving a set of boolean combinations of constraints is in general NP-complete, even if all the constraints have the form "$X=0$" or "$X=1$", unless the boolean combinations have very restricted form. (This is just the satisfiability problem. [Garey and Johnson, 79]) Solving a set of algebraic relations is NP-hard, and solving a set of transcendental equations is uncomputable, as discussed above.

6.2. Useful Techniques

Order relationships are almost always represented as DAG's. DAG's are actually constraint inference systems, since they infer the constraint $x>z$ from the constraints $x>y$ and $y>z$. A wholesale system can calculate the transitive closure of a DAG in time $O(n^3)$, where n is the number of nodes. Subsequent queries will require time $O(1)$. An incremental system can maintain the transitive closure in time at most $O(n^2)$ per input constraint, and then require time $O(1)$ for query answering. Alternatively, it can do no inference at assimilation time, allowing $O(1)$ assimilation, and then use an $O(n^2)$ algorithm at query time.

The best algorithms known which solve systems of difference relations can actually be viewed as carrying out the Waltz algorithm. See section 8 and appendix A for details.

There is an enormous mathematical literature dealing with the remaining problems: evaluating complex expressions over intervals, and solving systems of constraints. I am not competent to review this literature nor would it fall into the scope of this paper. The most important mathematical areas for these problems are interval analysis and optimization theory; for an introduction to these, see [Alefeld and Herzberger, 83] and [Leunberger, 73]. I will here only mention only those techniques which have been used or proposed for use in AI systems:

The Shostak SUP-INF algorithm is a constraint inference system for linear constraints [Shostak, 77]. It has been extended in in Rodney Brooks' CMS system, a subset of the ACRONYM vision system, to handle a certain class of non-linear constraints [Brooks, 81]. This system is quite powerful but slow, and its running time declines rapidly as the constraint set grows larger. Similar but simpler constraint inference systems were used by Ambler and Popplestone [Ambler and Popplestone, 75], Lozano-Perez [Lozano-Perez, 76] and Taylor [Taylor, 76]. Taylor's system finishes its operation by linearizing the simplified system of constraints and solving the linear inequalities using the simplex method. Use of the simplex method is also advocated by Malik and Binford [Malik and Binford, 83] for solving systems of linear inequalities. All of these techniques have the advantage of being deductive; the intervals they output are a superset of the true intervals.

Constraint inference systems are particularly powerful if many of the constraints involved are equations, rather than inequalities. Kuipers' constraint inference system [Kuipers, 85] propagates order relationships through addition, multiplication, and gate equations. A typical rule here is "If $A_1<A_2$, $B_1<B_2$, $C_1=A_1+B_1$, $C_2=A_2+B_2$, then $C_1<C_2$". Here the order relations on the A's and B's are propagated through the relation $C=A+B$. By using order relations with fixed quantities, the system is capable of doing interval propagation. The EL and ARS systems [Sussman and Stallman, 75], [Stallman and Sussman, 77] used propagation of algebraic terms around equations. Quantities are evaluated as algebraic expressions over other, basic quantities, When a quantity is assigned two different values, the values are equated and the algebraic equation solved.

The SPAM system [McDermott and Davis, 82] used a combination of Monte Carlo and hill-climbing techniques for term evaluation and refinement. The MERCATOR system [Davis, 84] used Monte Carlo techniques for term evaluation and special-purpose routines for refinement. Monte Carlo techniques are easy to program, run quickly, and are almost indifferent to the formal complexity of the term being evaluated. However, their accuracy declines rapidly with the number of quantities in the term, unless an exponential number of points are evaluated. Also, they are not useful for refinement, except in very special situations. Hill climbers are complicated to program and tend to run quite slowly, but they

are usually much more accurate than Monte Carlo. Neither hill climbers nor Monte Carlo can be depended on to be deductive. Monte Carlo search always gives a partial subset of the actual range; hill climbers often do.

Special purpose functions for individual terms and constraints are, by definition, efficient. They are generally deductively sound. They can be used for evaluating quite general terms by composing them, though composition often gives much too large a range. (See [Alefeld and Herzberg, 83, chap. 3] for an extensive discussion.) Their use for refinement is much more limited; refinement rules exist for linear constraints and for particularly simple non-linear constraints. (See figures 4 and 5) Their use in non-linear systems is therefore limited to cases where the nature of the domain causes a particular class of simple relations to appear frequently.

7. Propagating Signs

We now return to the consideration of specific constraint network systems, organizing them by label language and constraint language. We will first consider the simpler label language of signs and then the language of interval bounds.

For the purposes of sign computation, any function or constraint can be reduced to a finite table or relation over the sign of its arguments. All constraint languages thus look pretty much the same to a network with sign labels. Figure 6 gives the tables for some common functions and relations.

Complex arithmetic expressions can be evaluated by combining evaluation rules. For instance, if POS(x), POS(y), NEG(z), and NEG(w) then we can evaluate the expression $xy+wz$ by first evaluating the products as positive and then evaluating the sum as positive. They can be refined by evaluating all other parts of the relation and applying the refinement rules hierarchically. For example, given the relation $xy+wx(t+z)=-6$, we can evaluate xy as POS and -6 as NEG, giving $wx(t+z)$ as NEG. We next evaluate wx as NEG giving $t+z$ as POS. Finally, we use sum refinement to deduce that, since z is NEG, t is POS.

When sign labels are used, the Waltz algorithm is guaranteed to quiesce quickly. Since each quantity can change its value only once (from IND to a sign) in the course of execution, it can only once be responsible for putting a constraint on the constraint queue. The number of times a single constraint is put on the queue is thus L, the number of variables in the constraint. Summing over all constraints, the total number of times that some constraint is put on the queue is E, the size of the constraint set, the sum of the lengths of the constraints. The maximal number of calls to REFINE is thus EL.

If, as is often the case, a refinement rule can be applied only when only one parameter is unknown, there is a particularly efficient implementation. With each constraint, keep a count field of the number of unknown variables, which is updated each time a variable is set, and maintain a data structure which allows quick access to those constraints with count field 1. Always choose a constraint with count field 1 to refine; if there are none, then terminate. In this way, you never look at any constraint twice with the idea of refinement. Thus, quiescence is reached after e calls to REFINE, where e is the number of constraints. The overall running time of the algorithm is linear in E. (This is similar to the use of count fields in [McAllester, 80])

Propagation of sign labels is not complete overall for any interesting class of constraints. It is complete for assimilation for pure order constraints, but not for any more complex class of constraints. Nor is this surprising; the language of signs is simply too crude to capture much of what is going on in constraints of this kind. All that is available to a reasoner which uses signs are the tables of sign relationships implied by the constraints, as discussed above. The most we could expect is that the algorithm might be complete for the sign tables of some interesting class of constraints. It is, in fact, complete for the tables of bounded difference constraints. That is, given a collection of constraints on signs of the form in figure 6, and a number of starting sign labels, the Waltz algorithm gives the correct range

Linear relation:

Constraint: $\sum_{i \epsilon PC} c_i x_i - \sum_{i \epsilon NC} c_i x_i \epsilon [p,q]$;

/* PC, NC are, respectively, the sets of positive and negative coefficients. The c_i are assumed to be positive. */

Labels: $x_i \epsilon [a_i, b_i]$

Refinement:

For $j \epsilon PC$, $x_j \epsilon [\max(a_j, \frac{1}{c_j}(p + \sum_{i \epsilon NC} c_i a_i - \sum_{i \epsilon PC, i \neq j} c_i b_i)), \min(b_j, \frac{1}{c_j}(q + \sum_{i \epsilon NC} c_i b_i - \sum_{i \epsilon PC, i \neq j} c_i a_i))]$

For $j \epsilon NC$, $x_j \epsilon [\max(a_j, \frac{1}{c_j}(\sum_{i \epsilon PC} c_i a_i - q - \sum_{i \epsilon NC, i \neq j} c_i b_i)), \min(b_j, \frac{1}{c_j}(\sum_{i \epsilon PC} c_i b_i - p - \sum_{i \epsilon NC, i \neq j} c_i a_i))]$

Figure 4: Refinement over linear constraints

Constraint: $x^2 - yx + z = 0$

Labels: $x \epsilon [x_l, x_h], y \epsilon [y_l, y_h], z \epsilon [z_l, z_h]$

Refinement on x:

If $y_l > 0$ then let $a_l = y_l, a_h = y_h$,
else if $y_h < 0$ then let $a_l = -y_h, a_h = -y_l$,
else let $a_l = 0, a_h = \max(-y_l, y_h)$

/* a_l, a_h are the lower and upper bounds on abs(y) */

Let $D_l = \max(a_l^2 - 4z_h, 0), D_h = a_h^2 - 4z_l$.
If $D_h < 0$, then the system is inconsistent.

/* D_l, D_h are bounds on $y^2 - 4z$. */

Let $v_1 = 1/2(-y_h - \sqrt{D_h}), v_2 = 1/2(-y_l - \sqrt{D_l})$,

$v_3 = 1/2(-y_h + \sqrt{D_l}), v_4 = 1/2(-y_l + \sqrt{D_h})$

If $x_l > v_2$ then $x \epsilon [\max(x_l, v_3), \min(x_h, v_4)]$
else if $x_h < v_3$ then $x \epsilon [\max(x_l, v_1), \min(x_h, v_2)]$
else $x \epsilon [\max(x_l, v_1), \min(x_h, v_4)]$

/* As always, if the final interval for x has its lower bound greater than its upper bound, then the system is inconsistent */

Figure 5

Refinement rule for a non-linear constraint (used in the MERCATOR system [Davis, 84])

Order relations

Evaluation: X < Y if [NEG (X) and (ZERO (Y) or POS (Y))] or
[POS (Y) and (ZERO (X) or NEG (X))]

Refinement: X < Y implies
If (NEG (Y) or ZERO (Y)) then NEG (X).
If (POS (X) or ZERO (X)) then POS (Y).

Arithmetic Expressions

+	NEG	ZERO	POS	IND
NEG	NEG	NEG	IND	IND
ZERO	NEG	ZERO	POS	IND
POS	IND	POS	POS	IND
IND	IND	IND	IND	IND

*	NEG	ZERO	POS	IND
NEG	POS	ZERO	NEG	IND
ZERO	ZERO	ZERO	ZERO	ZERO
POS	NEG	ZERO	POS	IND
IND	IND	ZERO	IND	IND

X	NEG	ZERO	POS	IND
-X	POS	ZERO	NEG	IND

X	NEG	ZERO	POS	IND
1/X	NEG	UND	POS	IND

Constraint: X - Y >= C

C > 0 : either POS(X) or NEG(Y)
C = 0 : either POS(X) or NEG(Y) or (ZERO(X) and ZERO(Y))
C < 0 : no constraint on the signs of X or Y.

Figure 6
Sign Tables for Simple Functions and Relations

possible to each quantity. For any more complex system of constraints, the assimilation problem is NP-complete, so a quick algorithm like the Waltz algorithm has no chance of being complete. To get around this incompleteness, de Kleer and Brown [DeKleer and Brown, 85] use "generate and test" techniques, in which they try assigning particular signs to indeterminate quantities, and sees whether these assignments lead to contradiction. However, in view of the NP-completeness of the problem as a whole, this technique is clearly exponentially explosive when applied to large systems.

8. Propagating Intervals

The properties of label inference vary widely across the different kinds of constraints. Throughout the following discussion, n represents the number of quantities, e, the number of constraints, and E, the total size of the constraint system.

If the constraints are order relationships then the Waltz algorithm is complete for assimilation. Note that, when the high bound of a variable is reset, its new value is one of the original high bounds on one of the other quantities, and likewise for the lower bound. Thus, each variable can take on at most $2n$ different labels over the course of propagation. Therefore, by the theorem of Mackworth and Freuder [Mackworth and Freuder, 85], the running time of the Waltz algorithm is $O(ne)$. Cases can be constructed where this worst case is achieved.

Bounds on quantity differences, of the form $x-y\in[a,b]$, are easily analyzed, because the problem can be mapped onto the well known problem of finding a minimum-cost path in a directed graph. The analysis is carried out in appendix A. The major results are as follows: If we use a network with nodes of the form $x-y$, then the Waltz algorithm is complete for the whole inference process (assimilation together with query answering). Moreover, if we perform refinement in the proper order, then, for consistent sets of constraints, the system reaches quiescence in time $O(n^3)$. If we use a network with nodes for quantities, rather than their differences, then the Waltz algorithm is complete for assimilation, though not for inference as a whole. For consistent starting states, if constraints are chosen in the right order, the Waltz algorithm terminates in time $O(n^3)$. If all the constraints and labels have a positive upper bound and a negative lower bound, then constraints can be chosen in an order which gives convergence in time $O(n^2)$. In either case, if the starting state is inconsistent, then the system will either detect the inconsistency by finding a lower bound greater than an upper bound, or go into an infinite loop. Such infinite loops can be detected by simply monitoring the system and stopping it when it has performed more than the maximum number of refinements needed for a consistent system. With either differences as nodes or simple quantities as nodes, there are pathological cases in which choosing nodes in a poor order causes the system to do exponential number of refinements before it quiesces. It is therefore important in these systems to choose the order of refinements carefully.

The analysis of unit coefficient constraints is substantially more complicated, and is carried out in appendix B. The results may be summarized as follows. Like systems of bounded differences, propagation around unit coefficient constraints necessarily quiesces if the starting state is consistent. If we use either a FIFO queue for constraints in the propagation algorithm, or we choose constraints to refine in a fixed sequential order, then the Waltz algorithm will quiesce in time $O(nE)$, where n is the number of variables and E is the size of the constraint system (the sum of the lengths of all the constraints). On the other hand, if constraints are chosen in a poor order, then the Waltz algorithm may take exponential time to quiesce, and if the starting state is inconsistent, then it may go into an infinite loop. Unlike systems of bounded differences, however, the Waltz algorithm is not complete for assimilation.

Once we get past unit coefficient inequalities, to arbitrary linear relations, or non-linear relations, the Waltz algorithm starts to break down as a techniques. Not only is it incomplete; it tends to go into infinite loops even for very well-behaved sets of constraints.

Consider, for example, the simple pair of equations $\{x=y,\ x=2y\}$ with the starting ranges $\{x\epsilon[0,100],\ y\epsilon[0,100]\}$. The system is consistent, with the unique solution $x=y=0$. However, the Waltz algorithm goes into an infinite loop. We begin by deducing from the second equation that $y\epsilon[0,50]$; then, from the first equation, that $x\epsilon[0,50]$; then that $y\epsilon[0,25]$; then that $x\epsilon[0,25]$ ... Indeed, for practically any system of arbitrary linear relations, or non-linear relations, there are starting labellings which go into infinite loops in this way.

Nonetheless, the good properties of label inference -- its locality, its simplicity, its production of useful intermediate results -- are too useful to give up. Instead, iterated label inference is used, but more or less arbitrary rules are used to terminate it. The termination criterion can be a simple time limit, or a cutoff on the number of times any single constraint is invoked, or a cutoff on the amount of change that is considered significant. (Ultimately, the machine will enforce this last cutoff by itself, when the change goes below floating point number accuracy.)

The analysis of unit coefficient constraints in appendix B suggests some heuristics that may be useful even for non-linear constraints. There is reason to believe that, in general, it is a good heuristic strategy to choose constraints off the queue of algorithm 1 in FIFO order, or in fixed sequential order, rather than, say, LIFO order or best-first order. The intuitive justification for this rule is that, in any situation, there will generally be a few best refinements to apply, and FIFO or fixed order ensures that these will be applied reasonably soon. The analysis of appendix B makes this more precise. It also supports a somewhat stronger rule: that if we have been n times through the queue and have not reached a quiescent state, then we should probably quit, since we are likely to be in a very long or infinite loop.

Figure 7 summarizes the results of sections 6, 7, and 8.

9. Complex Nodes

So far, we have considered mostly nodes which correspond to a simple quantity. For many applications, however, it is useful to use nodes which correspond to a complex term of some kind. Frequently it will happen that the input constraints restrict the value of particular kinds of terms strongly, but restrict the value of single quantities only weakly or not at all. For example, as described above, there are often cases in reasoning about dates that the length of the interval between two dates is known fairly well, while the absolute dates are known much more vaguely. In reasoning about quantities which characterize object positions, it is common to know bounds on the distances between objects, and the angles that they form, but not to know their coordinates in any fixed coordinate systems. In such cases, it is common to have nodes which correspond to the relevant terms, rather than to the simple quantities.

Nodes of complex terms must be connected to the basic quantities and vice versa. Thus, such networks have three levels: the quantities (unlabelled); the nodes, with interval labels; and the explicit constraints, connected to the nodes. (Figure 8)

However, there are a number of problems with such an approach. Unlike simple quantities, complex terms cannot be assigned values arbitrarily; their values are related by implicit constraints. For example, terms of the form $x_{ij}=x_j-x_i$ obey the constraint $x_{ij}+x_{jk}=x_{ik}$. Label inference should be carried out through these implicit constraints, as well as through the usual explicit constraints. To achieve the maximum possible inferential power, we would like to have a *complete* set of implicit constraints: that is, a set of implicit constraints such that, if all the constraints are satisfied for a given set of term values, then that set of values is, indeed, possible for the terms. However, finding a usable complete set of intrinsic constraints is often intractable, and sometimes uncomputable.

A second, related problem is that the representation of explicit constraints and of queried terms does not, in general, have a unique representation as a function over the node terms. Therefore, the value returned by a query depends critically on which node terms are

Operation / Constraint Language	Theoretical Bounds	Sign Propagation	Interval Propagation
Unary Predicates	Trivial	Trivial	Trivial
Order Relation	$O(n^3)$	Complete: O(E)	Complete: $O(n^3)$
Bounded Difference	$O(n^3)$	Complete: O(E)	Complete: $O(n^3)$
Unit Coefficients	As hard as Linear Programming	Incomplete: O(E)	Incomplete: O(En)
Linear Programming	$O(n^{3.5}E^2)$	Incomplete: O(EL)	May not quiesce
Algebraic Equations	Doubly exponential time	Incomplete: O(EL)	May not quiesce
Transcendental Equations	Unsolvable	Incomplete: O(EL)	May not quiesce

Times given for "Bounds" are best known times for complete solutions.
Times given for sign and label propagation are time to reach quiescence.

n = number of variables
E = size of constraint set (sum of lengths of constraints)
L = maximum length of any constraint

Figure 7

Summary of Results

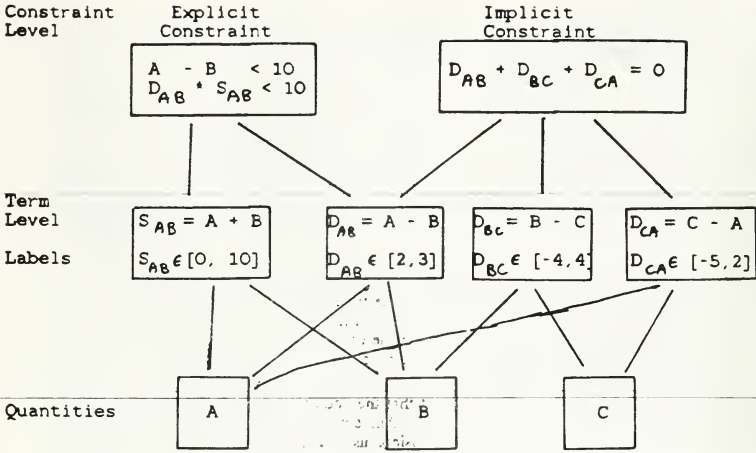


Figure 8

Three Level Constraint Network

used to compute it. Similarly, the effect of a input constraint and refinements which are deduced from it depend on how it is construed as a relation over nodes.

Suppose, for example, our system has interval labels for the quantities X_i, for terms of the form $D_{ij}=X_i-X_j$. and for terms of the form $S_{ij}=X_i+X_j$. We now query the system for the value of $T=X_1^2-X_2^2$. This can be evaluated, either as a function of X_1 and X_2, as in the above expression, or as the function $S_{12}D_{12}$. Suppose we start with the labels $X_1\epsilon[1,2]$, $X_2\epsilon[1,2]$, $D_{12}\epsilon[-1,1]$, $S_{12}\epsilon[2,4]$. Considering the term as a function over X_1 and X_2, we can calculate that its range is [-3, 3]. Considering it as a function over D_{12} and S_{12}, we can calculate its range as [-4, 4]. Here, the X nodes give better bounds than the complex nodes. However, if we change the label on D_{12} to be [0,0], then using $T=S_{12}D_{12}$, we can calculate $T\epsilon[0,0]$, while the calculation using the labels on X_1 and X_2 still gives [-3,3]. Thus, which expression is better depends on the starting states of the labellings.*

A final problem is that, even if we restrict terms to be of a fixed format, still there will be many more possible terms than simple quantities. If all of these must be represented in the system, we will waste a lot of space and inference time. There is no point in sustaining "date (IJCAI-85) - date (death (Caesar))" as a node term, and in reducing its range by a week when we get more precise information about the date of the conference. If we are to avoid combinatorial explosion, therefore, we must have rules which specify which nodes to use.

One way around these problems is to insist that the node terms be *independent*; that is, that any set of value assigned to the nodes (within some standard range) is internally consistent. In such a system, there are no intrinsic constraints; any term or relation has a unique dependence on the nodes; and the number of nodes is at most equal to the number of quantities. For example, in the case of difference relations, we could use X_0 and $D_{i0}=X_i-X_0$ as our node terms. Each of these can be assigned a value independently of all the others, and any quantity or term has a unique expression in terms of these nodes. $X_1^2-X_2^2$ must be evaluated as the function $(X_0+D_{10})^2-(X_0+D_{20})^2$. Other sets of independent constraints are possible; for example, the set $\{X_0,D_{i,i-1}$ for $i>0\}$. Indeed, any set of nodes, $\{X_0,D_{kj}\}$ forms an independent set, if it satisfies the following condition: Consider the quantities X_j as nodes and D_{kj} as edges connecting X_k to X_j. Then the set of D's must form a tree. This was the basic idea behind the use of "frob trees" in the SPAM system [McDermott and Davis, 84]. If an arbitrary root is chosen for the root, then the problem of finding node terms which connect two quantities can be solved quickly. Once these terms are found, it is easy to express a given expression over the quantities in terms of the connecting nodes.

The price of this simplicity that the problem of deciding which terms are represented as node becomes very insistent, since there is no room for redundancy. For example, suppose we are using the independent set $\{X_0,D_{i0}\}$, as above, and new bounds are input for X_2-X_1. There are two possible approaches. The first is to express X_2-X_1 as $D_{20}-D_{10}$, and use the new constraint to refine D_{20} and D_{10}. This, however, may lead to substantial loss of information. If we have the labels $D_{20}\epsilon[-3,5]$, $D_{10}\epsilon[-5,3]$ and we add the constraint $X_2-X_1\epsilon[-1,1]$, then refinement only brings our labels down to $D_{20}\epsilon[-3,4]$, $D_{10}\epsilon[-4,3]$. If we now ask for the value of X_2-X_1 from these labels, we evaluate that as $D_{20}-D_{10}$ and report that its range is [-7, 7]. Clearly, we have lost most of the information which was in the constraint.

The alternative is to add the term $D_{21}=X_2-X_1$ as a node. In order to maintain the independence of the node set, we must now take out one of the old nodes, either D_{20} or D_{10}.

* This problem can arise computationally with nodes which are simple labels, since it may not be clear how the algebraic expression can be simplified. (It may not even be computable) However, in principle, the functional dependence of a term on basic quantities is unique; the problem is merely one of computing this dependence over interval labels.

Whichever we do, we will end up sacrificing some information, and one has to judge that the information being added is worth the information being deleted. This process of adding some nodes and deleting others is examined extensively for spatial parameters in [Davis, 80], where it was called "remapping".

In the case above, the choice of which nodes to include is quite straightforward; a bound from -1 to 1 is worth more than a bound from -3 to 4. So we may sacrifice either D_{20} or D_{10} with a clear conscience. One way to express this choice criterion is to associate a numeric measure of uncertainty with a set of labels on a set of terms. For terms of the form X_i and D_{ij}, a plausible measure function can be obtained as follows: if the label on a node is $[L,U]$, then the measure of uncertainty on that node is $U-L$. By multiplying together the measures for the labels of all the nodes, one obtains a measure on the whole set of nodes. This measure has a natural interpretation as volume in the space R^{k+1} of the set of values of $X_0,X_1, \cdots X_k$ which satisfy all the labels on all the nodes. [Davis, 80]

However, with more complicated sets of node sets, the problem is much less clear cut. In particular, it is shown in [Davis, 80] that if one includes nodes of the form $F_{ijkl}=\frac{D_{ij}}{D_{kl}}$ then there is no coherent way to assign a numeric measure to the "amount of information" in a set of node labels; all that can be done is to define a partial ordering on the amount of information. Therefore, it may be ambiguous as to which set of nodes and labels is better.

Another problem which arises in remapping is the fact that it is sometimes advantageous to introduce new "imaginary" quantities. Suppose that we have the set of constraints $\{X_1-X_0\epsilon[-2,2],\ X_2-X_1\epsilon[-2,2],\ X_0-X_2\epsilon[-2,2]\}$. If we try to express these using an independent set drawn from the D_{ij}, we will inevitably lose information. If we record $D_{10}\epsilon[-2,2]$ and $D_{21}\epsilon[-2,2]$, then evaluating $X_0-X_2=-D_{21}-D_{10}$ will give us [-4,4]. However, we can express this constraint set exactly using a spare variable Y, and using the node terms $E_i=Y-X_i$. The label set $E_0\epsilon[-1,1]$, $E_1\epsilon[-1,1]$, $E_2\epsilon[-1,1]$ allows the retrieval of exactly the correct bounds on X_i-X_j. The quantity Y here is essentially a Skolem constant. I have not been able to analyse the problem of finding the best tree, given the ability to add imaginary quantities.

In short, the use of an independent set in general involves some loss of information, and, in the worst cases, very substantial loss of information. If we drop the requirement of a independent set, then we need not lose information, but we must confront the problems of choosing which nodes are worth keeping, and of choosing a representation for constraints and terms. Since the representation is not unique, the choice of the best representation generally involves some search at run-time. Usually, domain-dependent considerations are critical for these problems. A number of general observations are worth noting:

The kinds of terms used should be few, mathematically simple, and meaningful in the domain. For example, Dean's Time Map Maintainer [Dean, 85] uses differences of dates. SPAM [McDermott and Davis, 84] uses the relative position, orientation, and scale of two frames of reference. MERCATOR [Davis, 84] uses distances and orientations of line segments in the plane. (The underlying basic quantities in SPAM are the parameters of each reference frame with a respect to a single, fixed reference frame; in MERCATOR, the coordinates of the endpoints with respect to an absolute coordinate system.)

Networks are frequently organized hierarchically, where the elements of the hierarchy are single quantities or sets of quantities. For example, SPAM organizes each class of parameters (scale, orientation, position) into a more or less arbitrary tree. MERCATOR associates boundary points with the objects they describe, and organizes objects hierarchically by physical containment. Allen [Allen, 83] suggests the use of "reference intervals" to organize date lines, and Kahn and Gorry [Kahn and Gorry, 77] uses special reference dates in the same way. The hierarchy can be used to restrict the nodes in the system, by insisting that nodes related only parents and children, or possibly siblings, in the hierarchy. It can also be used to guide search, as discussed below.

The strict hierarchy can be extended by the use of a "kernel" set: a relatively small set of quantities which are in the focus of attention of the inference system. Within the kernel set, all term nodes are represented. Quantities are swapped into and out of the kernel as the system gains and loses interest in them. When a quantity is swapped out, only those terms which properly respect the hierarchy are preserved. Kernel sets have been used in Dean's TMM. [Dean, 85]

When a constraint is added or a term is evaluated, it must be expressed as a function over a set of independent nodes. Figure 9 shows a typical example from the MERCATOR program [Davis, 84]. The major problem here is the search problem of finding a set of nodes connecting the quantities of interest. If a network is organized hierarchically, and terms are kept which relate quantities which are father-and-son or siblings in the hierarchy, then relatively efficient strategies can often be found using a GPS-like strategy. Given two quantities P and Q which must be connected, their common ancestor in the tree, A, is located. At one level down in the hierarchy from A, we connect the ancestor of P to the ancestor of Q; and then, recursively, we connect P to its ancestor and Q to its ancestor. (Figure 10).

10. Propagation and Deletion in Incremental Systems

Two final problems remain in relation to large incremental systems. The first problem is the question of how far label inference should be taken when a new constraint is added to the system. In large networks, the complete Waltz algorithm may run very slowly, even for well behaved types of constraints. Three possibilities suggest themselves:

1) Propagate to quiescence and hope for the best. Generally, it is safe to assume that the effects of a single new constraint will only propagate through a small portion of the network. New information about the date of Hammurabi is unlikely to affect your belief about the date of the next Superbowl.

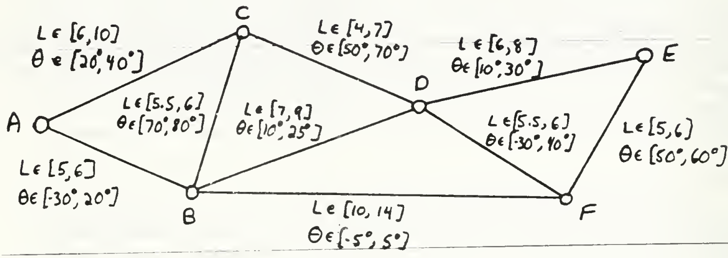
2) Don't propagate at all. Refine the arguments of the new constraint once, and never refer to it again. Note that in this approach, an explicit constraint need not be stored at all, since it is only used once. This approach makes the behavior of the system strongly dependent on the order in which constraints are added, but it runs quickly and is often effective enough. It was used in SPAM, because the refinement algorithm used a hill-climber, and therefore ran so slowly that one refinement was all that could be afforded.

3) Use some more or less arbitrary criterion (number of constraints refined on, elapsed time, etc.) to stop propagation after a while. As discussed above, this approach makes trouble unless the control structure is tight, since propagation may be resumed accidentally.

The other problem involves deletion. Ideally, in an incremental system, one would like to be able to delete constraints as well as to add them. Constraints are deleted when, temporally, they cease to be true, or when, inferentially, the knowledge base ceases to believe them. Generally, deletion is performed in networks using data dependencies, which record the connection between the conclusion of an inference and its premises. [Doyle, 79]. Data dependencies work well when there are many premises and relatively shallow inference. The success of this scheme in constraint networks therefore depends critically on the presumption that any constraint will have only local effects, even after more constraints have been added and propagated. However, it is hard to believe that this will be true in general; it would seem that, when enough time has passed after adding a constraint, almost all the labels will depend on it. If so, then deleting a constraint will involve recomputing all labels, starting from earlier values, and using all the remaining constraints. This is essentially a wholesale computation; the incremental features of the system are lost. If heuristic techniques

* This is based on my personal experience in spatial reasoning systems. Tom Dean informs me that, in temporal reasoning, such interactions are relatively slight. (Dean, personal communication)

The MERCATOR program records interval labels for the lengths and orientations of edges connecting vertices. A typical example is shown below:



Suppose we now query the system for the angle ACE. To evaluate this, MERCATOR must find a tree of edges connecting A, C, and E. The angle ∠ ACE is a unique function of the lengths and orientations of the edges in that tree. A typical tree might be the set { A-B, B-C, B-D, D-F, F-

The angle ∠ACE can be computed using formulas like

```
"X-COOR(C) - X-COOR(A) =
      LENGTH(A-B) * COS(ORIENT(A-B)) + LENGTH(B-C) * COS(ORIENT(B-C))"
```

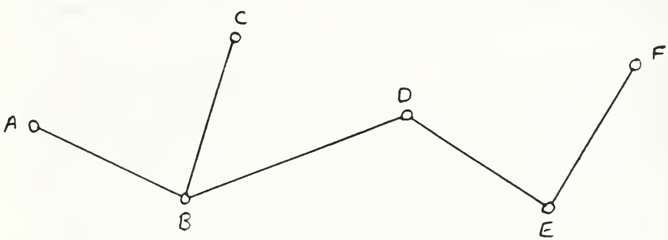
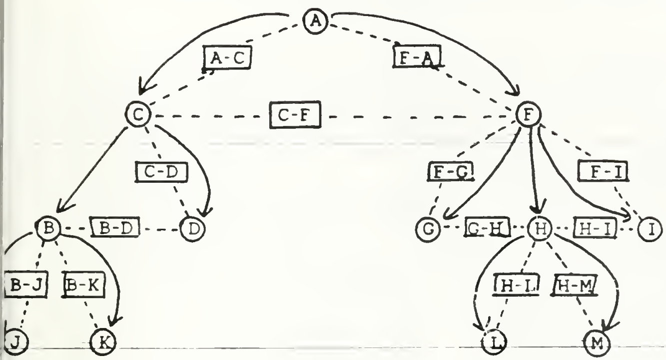


Figure 9

An Independent Set of Complex Nodes in MERCATOR

Hierarchical Constraint Network



Key: Circles are quantities. Rectangles are terms
Dotted lines connect quantities to terms.
Solid arrows indicate hierarchy of quantities.

To evaluate "L-J" as a function of the node terms, we first search from L and J upward in the tree until reaching the common ancestor A. At one level below that, we find C and F as the ancestors of L and J, and we connect C and F by the term "C-F". We must now connect C to J. Their common ancestor is C. One level down is B. B can be connected to C by going through D and using the two terms "B-D" and "C-D". B is connected to J by the single term "B-J". Connecting L and F is similar. Note that if the system contained a single term, "L-J", this search technique would miss it.

Figure 10

GPS Search in a Hierarchical Constraint Network

are used in systems with complex term nodes, then the problem is even more difficult, since choices as to how to expres a constraint or which nodes to include may have depended on label values which are now invalidated.

11. Conclusions

AI systems, when they can be analysed at all in terms of computational complexity, occupy a strange no-mans-land. The problems (game playing, theorem proving, scheduling, inductive inference) are known to be intractable in general, but AI systems must solve them for large, complex knowledge bases in close to constant time. To close this gap, we must either determine that the problems being addressed have special features that make them tractable, or we must be able to make do with approximate answers in a way which leads both to usable results and to fast algorithms.

In our discussion we have found two important parts of such an analysis. Firstly, there is a significant class of problems which can be expressed in terms of order relationships and bounded differences. Such sets of relationships can be completely solved for wholesale systems in polynomial time. Secondly, most applications of quantitative inference support a locality assumption that there is a natural grouping of quantities and that most constraints and terms relate quantities within the same grouping. It is plausible that an incremental system with such a locality property will lend itself to local algorithms, such as constraint propagation, and to a hierarchical organization; and these algorithms will run effectively and effectively on such a structure. However, no analysis has been performed which supports this presumption by a formal proof.

Beyond these two limited results, our analysis has little that is encouraging. As the language of constraints is made more complex, the inherent computational complexity increases rapidly, and the effectiveness and efficiency of propagating sign or interval labels declines rapidly. There is no analysis of the partial results produced by constraint propagation in these cases to tell us how they fail, and whether their success is adequate for the purposes of AI systems.

Analysis having failed, we may fall back on empirical evaluation; how well do these systems work in practice? Here the evidence is more encouraging. Kuipers [Kuipers, 85], Dean [Dean, 85], and de Kleer and Brown [deKleer and Brown, 85] report, of their various systems, that in practice they run quickly and effectively on problems of substantial size. Apparently, however, the code must be written with considerable care to achieve this [Dean, 85 and Kuipers, personal communication]. The major exceptions here have been the spatial reasoning systems SPAM and MERCATOR, both of which were extremely slow and fragile. (Quantitative inference was by no means the only or even the major problem in these systems; however, it was a substantial problem.) Spatial reasoning, of course, involves substantially more complex constraints than those used in the other systems mentioned.

The major outstanding research problems, therefore, seem to be the following:

1. Design an incremental system for quantitative inference in the spatial domain.
2. Determine the scope of such successful techniques as those cited above. Apply them to other domains.
3. Determine the power of systems like Kuipers' [Kuipers, 85], which propagate a simple binary constraint through more complex constraints.
4. Demonstrate analytically a relation between a locality assumption and the successful running of an incremental system.
5. Design an incremental system which allows for the effective deletion of constraints.

12. Appendix A: Propagation around Constraints of Bounded Differences

In this appendix, we consider constraint propagation around constraints of the form $x-y\epsilon[a,b]$. We first consider networks which contain as nodes all terms of the form $X_{ij}=X_j-X_i$, and then we consider networks with only nodes of the form X_i. In networks of the first kind, there are no explicit constraints; the constraint propagation occurs through implicit constraints of the form $X_{ij}+X_{jk}=X_{ik}$. There are two important results: (1) the Waltz algorithm is complete for assimilation in each of these systems; (2) if the system of constraints is consistent, then, given a proper order for refining constraints, the Waltz algorithm will terminate in time $O(n^3)$ in each of these systems.

The problem of propagating labels around a network of terms X_{ij} is closely related to the problem of calculating shortest paths on weighted graphs. Construct a graph whose nodes are the quantities x_i, and which has an arc of cost c_{ij} from x_i to x_j just if the constraint $x_j-x_i \leq c_{ij}$ is in the system. Then given any path $x_1,x_2 \cdots x_k$, with costs, $c_{12,}c_{23,} \cdots c_{n-1,n}$, these correpond to the inequalities $x_2-x_1 \leq c_{12}$, $x_3-x_2 \leq c_{23,} \cdots$. Summing these inequalities gives $x_n-x_1 \leq c_{12}+c_{23}+ \cdots +c_{n-1,n}$, Thus the relation $x_j-x_i \leq c$ is derivable from the equations iff there is a path from x_i to x_j of cost $\leq c$. Indeed, we can read the statement "$x_j-x_i \leq c$" as meaning just that, that there is a path from x_i to x_j of length at most c; in this case, the deduction is a deduction about paths in the graph. The system of equations is consistent just if there is no negative cost cycle in the graph. (The connection between the inequalities of this form and the shortest paths problem was first observed in [Pratt, 77].)

The basic refinement rule for bounded differences is as follows: If $X_{ij}\epsilon[a_{ij},b_{ij}]$, $X_{jk}\epsilon[a_{jk},b_{jk}]$, $X_{ik}\epsilon[a_{ik},b_{ik}]$ then X_{ik} may be refined to $X_{ik}\epsilon[\max(a_{ik},a_{ij}+a_{jk}),\min(b_{ik},b_{ij}+b_{jk})]$ This corresponds exactly to the central operation of most shortest path algorithms: $d(u,v)=\min(d(u,v),d(u,w)+d(w,v))$. Therefore, we can carry out these algorithms within a label inference system just by choosing the constraints to refine in the proper order. (The order, in other words, is all that the algorithm buys us.) In particular, we know that Floyd's algorithm finds all shortest paths in time $O(n^3)$. (See [Aho, Hopcroft, and Ullman, 74] p. 198) Therefore, we likewise can cause label inference to terminate in time $O(n^3)$ for consistent sets of constraints and labels by choosing the constraints to refine in the same order as Floyd's algorithm: First perform all refinement of constraints $a_{i1}+a_{1j}=a_{ij}$, then on constraints $a_{i2}+a_{2j}=a_{ij}$, and so on. Moreover, the correctness of Floyd's algorithm guarantees the completeness of the Waltz algorithm.

Recently, Fredman and Tarjan [Fredman and Tarjan, 84] have devised an improvement to Floyd's algorithm which runs in time $O(n^2\log n+en)$ where e is the number of edges (constraints), by using a carefully tailored data structure to choose the order in which to perform the above operation. Therefore, this running time can be achieved for Waltz's algorithm, at the cost of using a very complex control structure. On the other hand there known to be pathological examples where, by choosing to carry out the operations in the wrong order, it may take $O(2^n)$ operations to find the correct solution. (See, for example [Aho, Hocroft and Ullman, 74], p. 221 Problem 5.24). Therefore, the Waltz algorithm must be careful in choosing the order to do refinement, to avoid an unnecessary exponential running time.

If the constraints are inconsistent, then it is possible for the Waltz algorithm to enter into an arbitrarily long loop. Consider, for example, performing label inference around the constraints $x=y$ and $x=y+1$, starting with the labels $x\epsilon[0,1000]$, $y\epsilon[0,1000]$. We begin by using the second equation to deduce $x\epsilon[1,1000]$,$y\epsilon[0,999]$; then use the first equation to deduce $x\epsilon[1,999]$,$y\epsilon[1,999]$; then use the second equation to deduce $x\epsilon[2,999]$,$y\epsilon[1,998]$; and so on. It will take 500 iterations to discover the contradiction.

Systems which maintain just the simple quantities x_i as nodes can also be modelled in the same way. We can treat a bound on x_i as a bound on x_i-x_0, where x_0 is an additional quantity associated with the constant value 0. The problem is thus isomorphic to constructing single-source shortest paths algorithms, using only the assignments

"$d(x_0,x) \leftarrow \min(d(x_0,x), d(x_0,y)+d(y,x))$" If all arc lengths are positive -- that is, all upper bounds are positive and all lower bounds are negative -- then Dijkstra's algorithm gives an ordering of propagation which terminates in time $O(n^2)$ If there are negative arc lengths, then a cylic order of refinement, as in algorithm 2 below, gives an $O(n^3)$ solution.

```
/* This procedure computes the cost of the least cost path from 1 to i */

    procedure Single_Source (C : cost matrix)

    var D : array of reals;
/* D [i] is the cost of the cheapest path from 1 to i */

    begin for i := 1 to n do D[i] := C[0,i];
        for k := 1 to n-1 do
          for j := 1 to n do
            for i := 1 to n do
              D[j] := min (D[j], D[i] + C[i,j])
     end

/* At the beginning of the first iteration of the outer loop (k=1),
each vertex whose shortest path has one arc has D correctly set. At
the end of the first iteration, each vertex whose shortest path has
two arcs has D correctly set. At the end of the kth iteration,
each vertex whose shortest path has k arcs has D correctly set */
```

Algorithm 2

13. Appendix B: Interval Label Inference on Unit Coefficient Linear Inequalities

There are two major results in this appendix. The first deals with label inference around constraints of a general kind. It states that, for any set of interval refinements and any starting set of interval labels, one of two conditions holds. The refinements may contain a "self-dependency", in which the value of some bound depends, in a strong sense, on itself. This is a potential infinite loop. If there is no such self-dependency, then there is a "bee-line" set of refinements, which reaches quiescence after changing each bound on each variable at most once.

The second result applies to refinement on unit-coefficient constraints in particular. Here we can show that no consistent set of constraints can lead to a self-dependency, and that, therefore, there is always a bee-line set of refinements. We end with some discussion of how to apply these results algorithmically.

B.1: Dependencies and Redundancies in General Refinement

Definition 1: Given an n-tuple of variables $<X_1,X_2 \cdots X_n>$, a *valuation* is a function from the variables to reals. We write $V(X_i)=v_i$ or $V=<v_1 \cdots v_n>$.

Example: $<3,5,2>$ is a valuation on $<X_1,X_2,X_3>$

Definition 2: Given an n-tuple of variables $<X_1,X_2 \cdots X_n>$, a *labelling* is a function from the variables to closed intervals. We write $L(X_i)=[a_i,b_i]$ or $L=<[a_1,b_1], \cdots ,[a_n,b_n]>$ Note that a labelling L can be considered as a set of valuations on $<X_1 \cdots X_n>$, where a valuation V is in L if $V(X_i) \in L(X_i)$ for each i. Alternately, we can look at L as a valuation on the $2n$ variables $<X_1^l,X_1^u,X_2^l,X_2^u,...>$, where X_i^l, X_i^u are the upper and lower bounds of X_i. We will play off these two viewpoints in the course of the proof.

Example: $L = <[2.0, 4.0], [-1.0, 3.0], [5.0, 7.0]>$ is a labelling on $<X_1,X_2,X_3>$. If $V = <3.0, 0.0, 5.0>$ then $V \in L$. L is associated with the valuation $< 2.0, 4.0, -1.0, 3.0, 5.0,$

7.0 > on the variables $<X_1^l, X_1^u, X_2^l X_2^u, X_3^l, X_3^u>$.

Definition 3: If B is a bound on a variable X (i.e. B is either X^l or X^u) then SIGN(B) = 1 if B is an upper bound and -1 if B is a lower bound. If L and L' are labellings, L' is said to be *tighter* than L on B if $L'(B)=L(B)-SIGN(B)\cdot\Delta$ for some $\Delta>0$. Thus, an upper bound is tightened by lowering; a lower bound is tightened by raising.

Example: If $L = <$ [1.0, 3.0], [5.0, 7.0] $>$ and $L' = <$ [2.0, 3.0], [5.0, 6.0] $>$ are labellings on $<X_1, X_2>$, then L is tighter on X_1^l and L' is tighter on X_2^u.

Lemma 1: If L and L' are two labellings the following conditions are equivalent:

a) L' is a subset of L;

b) For each variable bound B, L' is tighter than or equal to L on B.

Proof: Immediate from Definition 3. In this case, we say that L' is at least as tight as L.

Example: < [2.0, 4.0], [5.0, 6.0] > is at least as tight as < [1.0, 4.0], [5.0, 7.0] >.

Definition 4: A *constraint* is a set of valuations. Given a constraint C, a labelling L,and a variable X we define REFINE(C,L,X_l) = $\{V(X_l) | V \epsilon L \cap C\}$. This is the set of values of X_l consistent with the labelling and the constraint.

Example: if $C = \{V \mid V(X_2) \geq V(X_1)\}$ and $L = <$ [1.0, 3.0], [0.0, 2.0] $>$, then REFINE(C,L,X_2) = [1.0, 2.0].

Definition 5: Given a constraint C and a variable X_l we define two *refinement operators* $R^u(X_l,C)$, and $R^l(X_l,C)$, which are functions from labellings to labellings. If $L=<a_1,b_1 \cdots a_n,b_n>$ then $R^u(X_l,C)(L)$ is formed by replacing b_l with upper_bound(REFINE(C,L,X_l)), and $R^l(X_l,C)(L)$ is formed by replacing a_l with lower_bound(REFINE(C,L,X_l)). Thus, these refinement operators allow us to replace one bound at a time.

Example: If C and L are as in the previous example, then $R^l(X_2,C)(L) = <$ [1.0, 3.0], [1.0, 2.0] $>$, and $R^u(X_2,C)(L) = <$ [1.0, 3.0], [0.0, 2.0] $>$.

Lemma 2: For any constraint C, variable X and labelling L,

a) $R^u(X,C)(L)$ and $R^l(X,C)(L)$ are subsets of L and supersets of $C \cap L$.

b) If L' is a subset of L then $R^u(X,C)(L')$ is a subset of $R^u(X,C)(L)$, and likewise for R^l.

Proof: Immediate from the definitions.

For any refinement operator R, the *output* bound of R, $OUT(R)$, is the bound which is affected by R. The *arguments* of R, $ARGS(R)$, are the bounds, other than $OUT(R)$ itself, which enter into the calculation of $OUT(R)$.

Example: Let C be the constraint $X_1 \geq X_3 + X_2 - 4.0$. $R = R^u(X_3,C)$ is the replacement of X_3^u by min (X_3^u, $X_1^u - X_2^l + 4.0$). Therefore $OUT(R) = X_3^u$ and $ARGS(R) = \{ X_1^u, X_2^l \}$.

Henceforward, we will be looking at sets of constraints and sets of refinement operators derived from those constraints. Given a series $R=<R_1,R_2 \cdots R_m>$ of refinements, and a labelling L, we will denote the composition of all the refinements as $R(L)=R_m(R_{m-1}(\cdots(R_2(R_1(L)))\cdots))$. A series S is a subseries of R if S contains a subset of the refinements in R in the same order. S need not be a consecutive subset; R_1,R_3,R_6 is a subseries of R_1,R_2,R_3,R_4,R_5,R_6. Given a series $<R_1 \cdots R_m>$, the notation $R_{l \ldots j}$ denotes the consecutive subseries $R_l,R_{l+1} \cdots R_j$. Series are assumed to be finite unless otherwise specified.

Lemma 3: Given a series R of refinement operators, if S is a subseries of R, then, for all labellings L, $R(L)$ is at least as tight as $S(L)$.

Proof by induction on the length k of S. If S has zero length, then $S(L)=L$ and $R(L)$ is tighter than L by lemma 2a above. If the statement is true for $S_{1 \ldots k-1}$, then let $S_k=R_j$. Since $S_{1 \ldots k-1}$ is a subseries of length $k-1$ of $R_{1 \ldots j-1}$, we know inductively that $R_{1 \ldots j-1}(L)$

is at least as tight as $S_{1 \ldots k-1}(L)$. Applying $S_k=R_j$ to this equation and using lemma 2b gives $R_{1 \ldots j}(L)$ is at least as tight as $S_{1 \ldots k}(L)$.

Definition 6: Given a series of refinement operators $R_1 \cdots R_m$ and a labelling L, R_i is said to be *active* if it changes the value of L; that is, $R_{1 \ldots i}(L) \neq R_{1 \ldots i-1}(L)$. If all the refinements in a series are active on L, the series is active on L.

Example: In the example of definitions 4 and 5, $R^u(X_2,C)$ is not active on L; $R^l(X_2,C)$ is active.

Lemma 4: Given a series R of refinement operators, and a labelling L, if Q is the subseries consisting of the active refinements, then $Q(L)=R(L)$.

Proof: Clearly, since inactive refinments do not affect the labelling, they may be omitted without changing the result.

Definition 7: Given a series of refinements $R_1 \cdots R_m$, we say that R_i is an immediate predecessor of R_j if $i<j$, $OUT(R_i) \in ARGS(R_j)$, and for all k such that $i<k<j$, $OUT(R_k) \neq OUT(R_i)$. Thus, some particular argument of R^j has been set most recently in the series by R_i. We say that R_j depends on R_i if (a) $j=i$ or (b) (recursively) R_j depends on R_k, and R_i is an immediate predecessor of R_k. We say that R_j depends on some bound B if, for some i, R_j depends on R_i and $B \in ARGS(R_i)$.

Example: Suppose $R=<R_1,R_2,R_3,R_4>$ and $OUT(R_1)=B_1$, $ARGS(R_1)=\{B_2,B_3\}$; $OUT(R_2)=B_2$, $ARGS(R_2)=\{B_4\}$; $OUT(R_3)=B_3$, $ARGS(R_3)=\{B_1,B_2\}$; $OUT(R_4)=B_1$, $ARGS(R_4)=\{B_2,B_3\}$. Then R_1 and R_2 are immediate predecessors of R_3; R_2 and R_3 are immediate predecessors of R_4; and R_4 depends (indirectly) on R_1. (See figure 11.)

Lemma 5: Let $R=<R_1 \cdots R_m>$ be a series of refinements, not all of which are dependent on R_1. Let $Q=<Q_1=R_1,Q_2 \cdots>$ be the refinements which depend on R_1 in order, and let $P=<P_1,P_2 \cdots P_k>$ be the refinements which do not depend on R_1. Then the series $R'=<P_1,P_2, \cdots P_k,Q_1,Q_2 \cdots>$ is at least as tight as R on any labelling L.

Proof: The P's do not depend on any of the Q's. Hence the value of the arguments of the P's are unchanged and they return the same values for their output variables. When each Q_i is entered, the input labelling is at least as tight as in the original series. Hence, the output variables are made at least as tight as they were in the original series. Thus, when we are all done, each variable has been set to a value at least as tight as its setting in the original series of refinements.

Example: In the example of Definition 7, the refinement series $R'=<R_2,R_1,R_3,R_4>$ is at least as tight as the series $R=<R_1,R_2,R_3,R_4>$, since R_2 does not depend on R_1. The reverse is not necessarily true, since, in R', R_1 does depend on R_2.

Definition 8: A series of refinements $R=<R_1 \cdots R_m>$ is *self-dependent* for labelling L if it is active on L and R_m depends on $OUT(R_m)$, its own output variable. *R contains a self-dependency* for L if, for some i and j, the sub-series $R_i \cdots R_j$ is self-dependent for $R_{1 \ldots i-1}(L)$.

Example: In the example of definition 7, if the series is active on some labelling L, then it is self-dependent on L since R_4 depends on $B_1=OUT(R_4)$.

Lemma 6: Any infinite sequence of active refinements contains a self-dependency for L.

Proof: For any N it is possible to choose refinements R_i, R_j, where $j>i>N$ and where R_i is an immediate predecessor of R_j. For if it were not possible, this would mean that for all $j>N$, all the arguments of R_j are set by refinement before N. This would mean that no refinement could be active twice after N, since it would get the same value as before. But in an infinite sequence of finitely many refinements, some refinements must appear infinitely often. By iterating this argument, we find that it is possible to find a subseries $Q=<Q_1,Q_2, \cdots Q_k>$ such that Q_i is an immediate predecessor of Q_{i+1} in R. Since this list

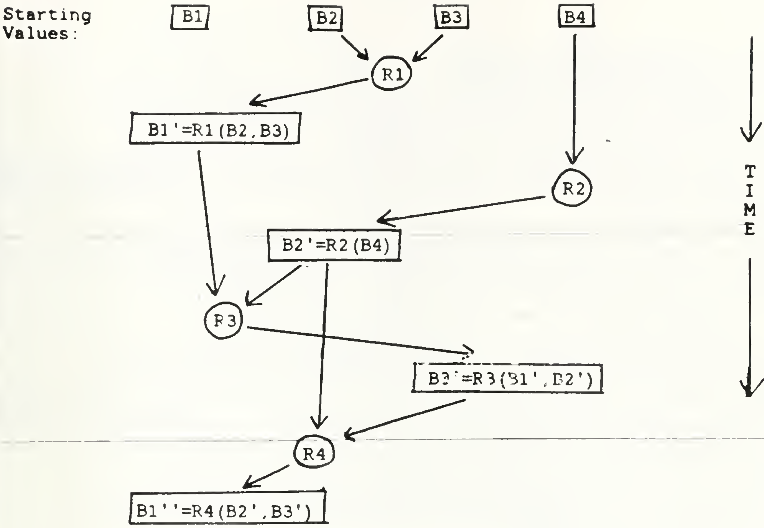


Figure 11

Dependency of Refinements

can be made arbitrarily long, at some point we will have two Q's with the same output bound. At this point, we have a self-dependency.

Definition 9: A series of refinement is *redundant* if two refinements have the same output variable bound. Example: the series of refinements in the example of definition 7 is redundant, since R_1 and R_4 have the same output variable B_1.

Lemma 7: Given a redundant series of active refinements $R=<R_1 \cdots R_m>$ and a labelling L, either the series contains a self-dependency for L or there exists a shorter series which at least as tight on L.

Proof: Assume, without loss of generality, that all the refinements are active on L. Since R is redundant, it contains multiple refinements with the same output variable. Let R_i and R_j, $i<j$, be two refinements with the same output variable such that no refinement in between i and j has that same output variable. There are two cases to consider:

1) R_j depends on R_i. In this case, the series contains a self-dependency.

2) R_j does not depend on R_i. Let $B=OUT(R_i)$ Consider the subseries $S=<R_i \cdots R_j>$. As in lemma 5, let $Q=<Q_1=R_i,Q_2 \cdots>$ be the refinements between R_i and R_j which do depend on R_i and let $P=<P_1,P_2, \cdots P_k=R_j>$ be those which do not depend on R_i. Then, by lemma 5 the series $S'=<P_1,P_2 \cdots P_k,Q_1,Q_2 \cdots>$ is at least as tight as S. Moreover, even if we remove R_i from S', leaving $S''=<P_1,P_2 \cdots P_k,Q_2,Q_3 \cdots>$, S'' is still as tight as S. For, in the original series S, R_j was active. Therefore, it set $OUT(R_j)$ to a tighter value than did R_i. Since it does not depend on any of the Q_i, it sets $OUT(R_j)$ to the same value in S'' as it did in S. The only effect of the inclusion of R_i in S', as opposed to its omission in S'', is that B may be set to a higher value still, but even without this it is still as tight as in S. But S'' is shorter than S. If we replace S'' for S in the original series R, we obtain a series shorter than R but at least as tight.

Example: Let $R=<R_1,R_2,R_3,R_4>$ where $OUT(R_1)=B_1$, $ARGS(R_1)=\{B_2,B_3\}$; $OUT(R_2)=B_2$, $ARGS(R_2)=\{B_3\}$; $OUT(R_3)=B_1$, $ARGS(R_3)=\{B_2,B_4\}$; $OUT(R_4)=B_5$, $ARGS(R_4)=\{B_1,B_2\}$. Suppose R is active on some labelling L. By Lemma 5, the modified series $R'=<R_2,R_3,R_1,R_4>$ is at least as tight on L, since neither R_2 nor R_3 depends on R_1. Moreover, the value calculated by R_3 for B_1 is unaffected by the transformation, since, in either series, R_3 depends only on the values of B_2, and B_4; and the value of B_2 calculated by R_2 depends only on the starting value of B_3. Thus, nothing is lost to R_3 by not having R_1 before it. If we now delete R_1 from R', leaving $R''=<R_2,R_3,R_4>$, the state of the labelling on entering R_4 is exactly the same as it was in R. Thus R'' is as tight as R, and shorter. (See figure 12.)

Definition 10: Given a set of refinements $S=\{R_1 \cdots R_k\}$, a labelling L is quiescent on the set if $R_1(L)=R_2(L)= \cdots =L$. Given a set of refinements $S=\{R_1 \cdots R_k\}$, a labelling L and a series of refinements from S, $Q=<Q_1,Q_2 \cdots Q_m>$, we say that Q brings L to quiescence if $Q(L)$ is quiescent on S.

Lemma 8: Given a set of refinements S and a labelling L, let P and Q be two series of refinements drawn from S. If $Q(L)$ is quiescent on S then it is at least as tight as $P(L)$.

Proof: Since $Q(L)$ is quiescent, $P(Q(L))=Q(L)$. (None of the refinements in P can have any effect on $Q(L)$.) However, since $Q(L)$ is at least as tight as L, $P(Q(L))$ is at least as tight as $P(L)$. Hence $Q(L)$ is at least as tight as $P(L)$.

Corollary: Given a set of refinements S and a labelling L, if Q and P are two different series of refinements both of which bring L to quiescence then $Q(L)=P(L)$.

Proof: By the above lemma, $P(L)$ is as tight as $Q(L)$ and vice versa, so the two are equal.

Theorem 1: Given a set of refinements, $S=\{R_1 \cdots R_m\}$, on n variables, and a labelling L, then either there exists a series of refinements in S containing a self-dependency on L, or

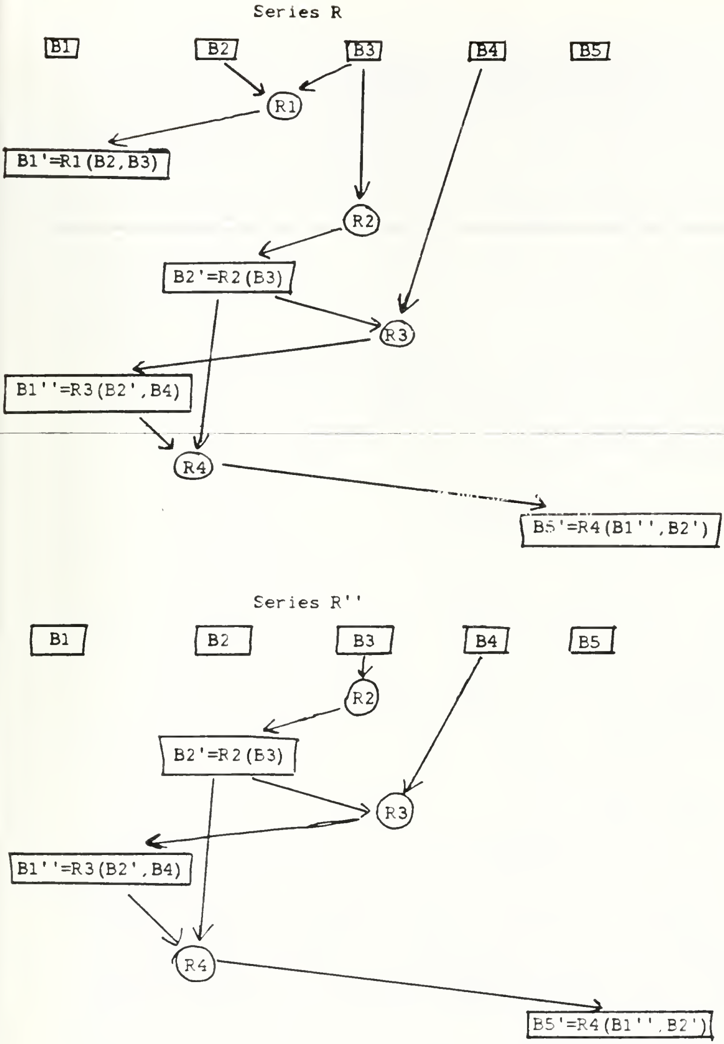


Figure 12

Compressing a Series of Refinements

there exists a series of refinements which is not redundant which brings L to quiescence.

Proof: Let Q be any infinite series of refinements from S in which each refinement appears infinitely often. Let R be the subseries of Q containing all the active refinements. If R has no self-dependency, then by lemma 6 it must be finite. If R is finite, then $R(L)$ must be quiescent on S, since, after the end of R, Q tries all the different refinements in S and none of them are active. Moreover, since R has no self-dependency, it can be gradually pruned down, step-by-step, using lemma 7 until it has no redundancies left. Each step leaves it at least as tight as it was before. Since no series of refinements can get tighter than the quiescent state (lemma 8) the resultant series of refinements must reach quiescence.

B.2: Unit Coefficient Constraints.

Definition 11: A *unit coefficient constraint* is a relation C of the form $p \leq \sum_{k \in PC(C)} X_k - \sum_{k \in NC(C)} X_k \leq q$, where p and q are constants and $PC(C)$ and $NC(C)$ are disjoint sets of variable indices. (PC and NC stand for "positive coefficients" and "negative coefficients".)

Henceforward in this section, we will assume that all constraints are unit coefficient constraints, and that all refinements are drawn from unit coefficient constraints.

Lemma 9: For any variable X_i, labelling L, and constraint C of the form in definition 11, REFINE(C,L,X_i) can be calculated as follows: Let $L(X_k)=[a_k,b_k]$ for all k and let REFINE$(C,L,X_i)=[a'_i,b'_i]$.

If $i \in PC(C)$ then

$$a'_i=\max(a_i, p+\sum_{k \in NC(C)} a_k - \sum_{k \in PC(C), k \neq i} b_k);$$

$$b'_i=\min(b_i, q+\sum_{k \in NC(C)} b_k - \sum_{k \in PC(C), k \neq i} a_k).$$

If $i \in NC(C)$ then

$$a'_i=\max(a_i, \sum_{k \in PC(C)} a_k - q - \sum_{k \in NC(C), k \neq i} b_k);$$

$$b'_i=\min(b_i, \sum_{k \in PC(C)} b_k - p - \sum_{k \in NC(C), k \neq i} a_k).$$

Proof by solving the inequality.

Lemma 10:

If refinement $R(X,C)$ is active on L then R has one of the following forms:

If $i \in PC(C)$ then

$$a'_i=p+\sum_{k \in NC(C)} a_k - \sum_{k \in PC(C), k \neq i} b_k;$$

$$b'_i=q+\sum_{k \in NC(C)} b_k - \sum_{k \in PC(C), k \neq i} a_k.$$

If $i \in NC(C)$ then

$$a'_i=\sum_{k \in PC(C)} a_k - q - \sum_{k \in NC(C), k \neq i} b_k;$$

$$b'_i=\sum_{k \in PC(C)} b_k - p - \sum_{k \in NC(C), k \neq i} a_k.$$

Proof: Immediate from Lemma 9.

Lemma 11: Let the refinement R be active on L. Let B be a bound in $ARGS(R)$. Let L' be tighter than L, and let $L'(B)$ be tighter than $L(B)$ by an amount $\Delta > 0$. Then $R(L')$ is tighter than $R(L)$ on $OUT(R)$ by at least Δ.

That is, under the specified conditions, a change in the input is mirrored by at least as great a change in the output. Here the limitation to unit coefficient equations is critical; this is not true of other kinds of refinement.

Proof: Since R is active on L, $R(L)$ evaluates $OUT(R)$ as the sum and difference of the bounds in $ARGS(R)$, including B. R is not necessarily active on L', since $OUT(R)$ may have

been very much tightened in L'. However, in computing $OUT(R)$ in $R(L')$ we make sure that it is at least as tight as the same sum and difference over the values in L'. Since tightening in the arguments shows up as tightening in the sum with coefficient 1, this sum is tighter in L' than in L by the sum of the tightenings of all the bounds in $ARGS(R)$. Hence, $OUT(R)$ is tighter in $R(L')$ than in $R(L)$ by at least the sum of all the tightenings. However, this sum contains no negative terms, and one term Δ. Hence, $R(L')$ is tighter than $R(L)$ on $OUT(R)$ by at least Δ.

Example: Let C be the constraint $4 \geq X_1 - X_2 + X_3$. Let $R = R^u(X_3, C)$. Thus R sets X_3^u to min $(X_3^u, -X_1^l + X_2^u + 4)$. Let L = < [5.0, 8.0], [2.0, 4.0], [0.0, 5.0] >, and L' = < [7.0, 8.0], [2.0, 4.0], [0.0, 2.0] >. Then L' is tighter by 2.0 on X_1^l, which is in $ARGS(OUT(R))$. $R(L)$ evaluates X_3^u as 3.0, and $R(L')$ evaluates it as 1.0. Thus $R(L)$ is tighter than $R(L')$ by 2.0 on X_3^u.

Lemma 12: Let the sequence $R = <R_1, \cdots, R_m>$ be active on a labelling L and dependent on the variable bound B. Let $B_m = OUT(R_m)$, the last variable bound set. Let L' be a labelling which is tighter than L, such that $L'(B) = L(B) - SIGN(B) \cdot \Delta$, for some $\Delta > 0$. Then $R(L')$ is tighter than $R(L)$ on B_m by at least Δ.

Proof: By lemma 2a, since L' is tighter than L, $R_{1 \ldots i}(L')$ is at least as tight as $R_{1 \ldots i}(L)$ for each i. Let $R_{i_1}, R_{i_2} \cdots$ be a subseries of R such that $B \in ARGS(R_{i_1})$, $B_1 = OUT(R_{i_1}) \in ARGS(R_{i_2})$ etc. Since no refinement has reset B_0 before applying R_{i_1}, it is still true that $R_{1 \ldots i_1 - 1}(L')$ is tighter than $R_{1 \ldots i_1 - 1}(L)$ on B by Δ. (These are the labellings just before applying R_{i_1}.) By lemma 11, it follows that $R_{1 \ldots i_1}(L')$ is tighter than $R_{1 \ldots i_1}(L')$ on B_1 by at least Δ. (These are the labellings after applying R_{i_1}.) Similarly, since B_1 is not reset before applying L_{i_2}, it follows that $R_{1 \ldots i_2 - 1}(L')$ is tighter than $R_{1 \ldots i_2 - 1}(L)$ on B_1 by Δ. Again, it follows that $R_{1 \ldots i_2}(L')$ is tighter than $R_{1 \ldots i_2}(L')$ on B_2 by at least Δ. And so it goes inductively, up through R_m.

Lemma 13: Let $R = <R_1 \cdots R_m>$ be self dependent for L. Let $B = OUT(R_m)$. Construct the infinite series of refinements $R^\infty = RRR.. = <R_1, R_2 \cdots R_m, R_1, R_2, \cdots R_m, R_1 \cdots>$ by iterating the series R infinitely often. Let $L_0 = L$, $L_1 = R(L)$, $L_2 = R^2(L) = R(R(L))$, etc. Then the sequence $L_0(B)$, $L_1(B)$, $L_2(B)$ ... increases or decreases at least arithmetically.

Proof: Since R_m is active, $R(L) = L_1$ is tighter than L on B. Let $\Delta = \text{abs}(L_1(B) - L(B))$. Then L_1 satisfies the conditions for L' in Lemma 12. Hence $R^2(L) = L_2$ is tighter than L_1 on B by at least Δ, and hence tighter than L by at least 2Δ. Using L_2 as L' in Lemma 12 gives the conclusion that L_3 is tighter than L_2 on B by at least Δ. Inductively, L_k is tighter than L on B by at least $k\Delta$.

Lemma 14: If $R_1 \cdots R_m$ is self-dependent for L, then the original set of constraints and labels was inconsistent.

Proof: We are able to construct a series of refinements which tightens B toward plus or minus infinity. This means either that some upper bound on some variable is arbitrarily small or that some lower bound is arbitrarily large, which is an impossible conclusion. However, since each refinement is a necessary consequence of the original state of the system, the original system must have been inconsistent.

Theorem 2: If $R_1 \cdots R_n$ contains a self-dependency for L then the original constraints together with L are inconsistent.

Proof: Let $R_i \cdots R_j$ be self-dependent. The original set of constraints together with L imply L_{i-1}. Using lemma 14 on L_{i-1} and $R_i \cdots R_j$ shows that the constraints and original labelling are inconsistent.

Theorem 3: Starting with a consistent set of constraints and a labelling L, there is a non-redundant series of refinements which reaches a quiescent state. This may be called a

"bee-line" series for L.

Proof: Follows immediately from theorems 1 and 2.

Corollary: Given a set of contraints on n variables and a labelling L which are consistent, let R be a series containing each refinement drawn from the set exactly once. Then $R^{2n} = <R, R, R, \cdots (2n \text{ times})>$ reaches quiescence.

Proof: R^{2n} contains the bee-line series as a subseries, and therefore is as tight as the bee-line series on L. Since the bee-line series reaches quiescence, so does R^{2n}.

The only remaining question is how quickly this can be implemented. It turns out that all the refinements for a single constraint on a labelling can be together in time O(length of the constraint), by incrementally changing the sum and differences in lemma 10, rather than recalculating each time. Thus, the average time to apply a refinement is O(1). Thus, we are repeating $2E$ refinements $2n$ times, giving us an $O(nE)$ algorithm.

The techniques of [Mackworth and Freuder,85], used in algorithm 1, can be applied to make this practically more efficient, if not to give theoretically better bounds. The idea is that if none of the arguments of a given constraint have changed since it was last refined, then the constraint need not be refined again. Thus, we can use a queueing structure, where a constraint is added to the queue when one of its arguments is changed. It is easily shown that, if constraints are taken off the queue, either in fixed, cyclic order or in FIFO order, then the series of refinements will be a superseries of the bee-line series when it is not more than E times as long, where E is the number of refinements. Thus, either ordering scheme gives us a worst case of $O(nE)$ running time before reaching quiescence.

Acknowledgements

Drew McDermott introduced me to the problems described here, and supervised much of my research on them. I would also like to thank Sanjaya Addanki, Philip Davis, and Tom Dean for their comments on an earlier draft. The work in this paper was supported in part by NSF grant DCR-8402309.

Bibliography

[Aho, Hopcroft, and Ullman, 74] Aho, A.V., Hopcroft, J., and Ullman, J., *The Design and Analysis of Computer Algorithms,* Addison-Wesley, 1974

[Alefeld and Herzberger, 83] Alefeld, G. and Herzberger, J., *Introduction to Interval Computations,* Academic Press, 1983

[Allen, 83] Allen, J.F. "Maintaining knowledge about temporal intervals", *Communications of the ACM,* Vol. 26 pp. 832-843, 1983

[Ambler and Poppelstone, 75] Ambler, A.P. and Popplestone, R.J., "Inferring the Positions of Bodies from Specified Spatial Relationships", *Artificial Intelligence Journal,* Vol. 6, pp. 175 - 208, 1975

[Boggess, 79] Boggess, L., "Computational Interpretation of English Spatial Prepositions", Report T-75, CSL, University of Illinois, 1979

[Brooks, 81] Brooks, R.A. "Symbolic Reasoning Among 3-D Models and 2-D Images", *Artificial Intelligence Journal,* vol. 17, pp 285-348, 1981

[Davis, 81] Davis, E., "Organizing Spatial Knowledge", Yale University Research Report #193, 1981

[Davis, 84] Davis, E., "Representing and Acquiring Geographic Knowledge", Yale University Research Report #292, 1984

[Dean, 85] Dean, T., "Planning and Temporal Reasoning Under Uncertainty", IEEE Conference on Knowledge Representation, 1984

[DeKleer and Brown, 85] DeKleer, J. and Brown, J.S. "A Qualitative Physics Based on Confluences", *Artificial Intelligence Journal,* vol. 24, pp. 7-83, 1985

[Doyle, 79] Doyle, J., "A Truth-Maintenance System", *Artificial Intelligence Journal*, Vol. 12, pp. 231-272, 1979

[Forbus, 85] Forbus, K., "Qualitative Process Theory", *Artificial Intelligence Journal*, vol. 24, pp. 85-168, 1985

[Fredman and Tarjan, 84] Fredman, M.L. and Tarjan, R., "Fibonacci Heaps and Their Usess in Improved Network Optimization Algorithms," *25th IEEE FOCS* pp. 338-346, 1984

[Freuder, 78] Freuder, E., "Synthesizing Constraint Expresions", *Communication of the ACM*, vol. 21 no. 11, pp. 958-966, 1978

[Garey and Johnson, 79] Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979

[Hummel and Zucker, 83] Hummel, R.A. and Zucker, S.W., "On the Foundations of Relaxation Labeling Processes", *IEEE Trans. PAMI* Vol. 5, pp. 267-287, 1983

[Kahn and Gorry, 77] Kahn, K. and Gorry, G.A., "Mechanizing Temporal Knowledge," *Artificial Intelligence Journal*, pp. 87-108, 1977

[Karmakar, 84] Karmakar, N., "A New Polynomial Time Algorithm for Linear Programming," *16th ACM STOC* pp. 302-311, 1984

[Kozen and Yap, 85] Kozen, D. and Yap, C.K., "Algebraic Cell Decomposition in NC," *26th IEEE FOCS*, pp. 515 - 521, 1985

[Kuipers, 85] Kuipers, B., "Commonsense Reasoning about Causality: Deriving Behavior from Structure", *Artificial Intelligence Journal*, vol. 24, pp. 169-203, 1985

[Lozano-Perez, 76] Lozano-Perez, T. "The design of a mechanical assembly system", Tech. Rep. AI-TR-397, MIT, 1976

[Luenberger, 73] Luenberger, D.G., *Linear and Non-Linear Programming*, Addison-Wesley, 1973

[Mackworth, 77] Mackworth, A.K., "Consistency in networks of relations," *Artificial Intelligence Journal*, Vol. 8, pp. 99-118, 1977

[Mackworth and Freuder, 85] Mackworth, A.K. and Freuder,E.C., "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems," *Artificial Intelligence Journal*, vol. 25, pp. 65-74, 1985

[Malik and Binford, 83] Malik, J. and Binford, T.O., "Reasoning in Time and Space," Proc. IJCAI-8, pp. 343-345, 1983

[McAllester, 80] McAllester, D. "An Outlook on Truth Maintenance," MIT AI Lab, Tech. Memo 521, 1980

[McDermott, 80] McDermott, D.V., "Spatial Inferences with Ground, Metric Formulas on Simple Objects", Yale University Research Report #173, 1980

[McDermott and Davis, 82] McDermott, D.V. and Davis, E., "Planning Routes Through Uncertain Territory," *Artificial Intelligence Journal*, vol. 22, pp. 107-156, 1984

[Newell and Simon, 72] Newell, A. and Simon, H.A., *Human Problem Solving*, Prentice-Hall, 1972

[Pratt, 77] Pratt, V.R., "Two easy theories whose combination is hard", Tech. Rep. MIT, 1977, cited in Shostak, R. "Deciding Linear Inequalities by Computing Loop Residues," *JACM*, Vol. 28, No. 4, pp. 769 - 779, 1981

[Richardson, 68] Richardson, D., "Some Undecidable Problems Involving Elementary Functions of a Real Variable," *Journal of Symbolic Logic*, Vol. 33, pp. 514 - 520, 1968

[Sacerdoti, 77] Sacerdoti, E., *A Structure for Plans and Behavior*, Elsevier, 1977

[Shostak, 77] Shostak, R.E. "On the sup-inf method for proving Presburger formulas," *Journal of the ACM*, Vol. 24, pp. 529-543, 1977

[Stallman and Sussman, 77] Stallman, R.M. and Sussman, G.J. "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence Journal*, Vol. 9, pp. 135-196, 1977

[Sussman and Stallman, 75] Sussman, G.J. and Stallman, R.M. "Heuristic Techniques in Computer-Aided Circuit Analysis", *IEEE Transactions on Circuits and Systems* CAS-22, 1975

[Sussman and Steele, 80] Sussman, G.J. and Steele, G.L., "CONSTRAINTS - A Language for Expressing Almost Hierarchical Descriptions", *Artificial Intelligence Journal*, Vol. 14, pp. 1-40, 1980

[Taylor, 76] Taylor, R.H. "A synthesis of manipulator control programs from task-level specifications", Memo AIM 282, Stanford University, 1976

[Vere, 83] Vere, S., "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Transactions on Patern Analysis and Machine Intelligence*, Vol. 5 No. 3, May 1983

[Waltz, 75] Waltz, D., "Understanding line drawings of scenes with shadows," in *The Psychology of Computer Vision*, P. Winston, ed., McGraw Hill, 1975

[Winograd, 71] Winograd, T. *Understanding Natural Language* Academic Press, 1972

[Yemini, 79] Yemini, Y., "Some Theoretical Aspects of Position-Location Problems," *Proc. 20th Symp. on the Foundations of Computer Science*, pp. 1-7, 1979

NYU COMPSCI TR-189 c.1
Davis, Ernest
Constraint propagation on
real-valued quantities.